Cs-501
Quiz no.3
Lecture 23 to 30

Sunrise Zeeshan

Subscribe My Youtube Channel

1. Along with information bits we add up another bit which its called the bit.
   CRC
   Hamming
   Error Detection
   Parity
2. Tri-state buffers are used for removing _____
   Instruction collision
   bus collision
   Instruction contention
   bus contention
3. In which technique does the hardware directly access host memory for reading or writing independent of CPU?
   Direct Memory Access (DMA)
   Programmed I/O
   Interrupt driven I/O
   Polling
4. The main issue/s in error control is/are _____
   Detection of Error
   Correction of Error
   Both Detection of Error and Correction of Error
   Avoidance of Error
5. The source file of FALS!IM should contain_____ text only.
   Unicode
   ASCII
   ANSI
   UTF
6. The directive_____ is used to define vanables.
   .equ

.org
.db
.sw

7. human works with base 10 and computers work with base_____
   8
   10
   2
   11
8. The information about interrupt vector is given in &-bits, from bit 0 to 7. which is translated to bit_____ on the data bus.
   16 to 23
   11 to 18
   0 to 7
   8 to 15
9. The_____ can be determined from the number of platters and the number of tracks.
   Speed of processing
   execution time
   storage capacity
   Latency
10. The conversion of numbers from 4 representation in one base to another is known as.
    Radix Conversion
    Number Representation
    Decimal representation
    Hexadecimal Representation
11. Every interrupt handler has an interrupt return (IRET) instruction, this instruction is an example of_____ return.
    NEAR
    FAR
    SHORT
    RELATIVE
12. A software routine performed when ani interrupt is received by the computer is called as ------—
    Interrupt
    Interrupt handler
    Exception
    Trap
13. Identify the following type of serial communication error condition: “The prior character that was received was not still read by the CPU and is over written by a new recetved character.”
    Framing error
    Overrun error
    Parity error

Under-run error

14. Taking control of the system bus for a few bus cycles is known as_____
    Bus Stealing
    Cycle Stealing
    Cycle Transfering
    None of given
15. What is the status of the ACKNLG# signal when a character is completely received by the printer?
    It. goes from low to high
    It goes from high to low
    It toggles its state
    lt remains unaffected
16. How can you define an interrupt?
    A process where an extemal device can spi e workinglof the microprocessor
    A process where memory can a a execution speed
    A process where an external device can get the attention of the microprocessor
    A process where input devices can takeover the working of the microprocessor
17. How Interrupt driven I/O is better than polling because?
    Interrupt driver I/O is easy to design
    Interrupt driver |/O is enki defi version of Polling
    Interrupt driver I/O does not waste time on checking which device is available.
    Interrupt driven I/O is easy to program.
18. For input ports, the incoming data should be placed on the data bus only during the I/O read bus cycle. For this Purpose_____ are used.
    Flip Flop
    Tri-state Buffers
    AND Gates
    Registers
19. In the little-endian format exchanging data between computers, the data transmitted by one will be received in a “swapped” form by the other.
    Organized
    Signals
    Swapped
    Arranged
20. Which |/O technique will be used by a sound card that may need to access data stored in the computer's RAM?
    Programmed I/O
    Interrupt driven I/O
    Direct memory access(DMA)
    Palling
21. Select the parts of a hard disk.
    Data section and a trailer

Header only
Data section only
Header, data section and a trailer.

22. -—-—--— allows a peripheral to read and write memory without intervention by the CPU.
    Programmed I/O
    Interrupt driven I/O
    Direct memory access(DMA)
    Polling
23. The Pentium does allow the use of some part of its accumulator register EAX
    8 bits
    16 bits
    32 bits
    64 bits
24. A component connected to the system bus and having control of it during a particular bus cycle is called_____
    Master component
    System bus
    Buffer component
25. In the litthe-endian format exchanging data between computers, the data transmitted by one will be received in a “swapped” form by the otr. he
    Organized
    Signals
    Swapped
    Arranged
26. A computer interface is an_____ circuit that matches the requirements of the two subsystems between which it is connected.
    Digital
    Electronic
    Primary
    Obituary
27. n which one of the following methods for resolving the priority, the device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the senes?
    Asynchronous
    Daisy-Chaining Priority
    Parallel
    Semi-synchronous
28. Every interrupt handler has an interrupt return (IRET) instruction, this instruction is an example of_____ retrun.
    Repeat

29. ET = ____
    CP x IC x T
    CPl x IC x T
    CPI/IC x T
    CPI x IC/T
30. In 8086/8088 processor, interrupt vector table is located at the memory location_____
    0
    4
    256
    1050
31. Every time you press a key, an interrupt is generated. This is an example of
    Hardware interrupt
    Software interrupt
    All of the given
    None of the given
32. Most parallel |/O ports used with peripheral devices are mapped on a range of_____
    Bus addresses
    Direct memory Access
    contiguous addresses
    Cache
33. Which ts the last instruction of the ISR that is to be executed when the ISR terminates?
    IRET
    IRQ
    INT
    NMI
34. Why DMA is faster than Programmer I/O technique because?
    DMA transfers data directly using CPU
    DMA transfers data directly using CPU
    DMA uses buffers with CPU
    DMA uses interrupted driven I/O
35. Given an m-digit base b number x, the of x is x =(b™—x)mod b™
    Radix Compliment
    Biased Representation
36. How does DMA saves CPU time?
    By controlling data transfer between l/O Device and memory directly.
    By periodically polling.
    By issuing an interrupt request to the CPU to request attention.
37. _____is the simplest form for representing a signed number.
    Biased Representation
    Diminished Radix Compliment Form
    Sign Magnitude Form
    None of the given

38. _____signal is used in printer with DB-25 interface to reset its controller.
    #PE
    #5TROB
    #INIT
    #SLCT
39. A component connected to the system bus and having control of it during 4 particular bus cycle is called_____
    Slave component
    Master component
    System bus
    Buffer component
40. In @ printer with DB-25 interface,_____ signal is better for edge triggered systems.
    BUSY#
    PE#
    ACKNLG#
    STROB#
41. _____ is a technique in which some of the CPU's address lines forming an input to the address decoder are ignored.
    Partial Decoding
42. An Interface that can be used to connect the microcomputer bus to _____ is called an I/O port.
    peripheral devices
43. Connection to a CPU that provides a data path between the CPU and external devices, such as a keyboard, display, or reader is called----------
    I/O port
44. In which one of the following methods for resolving the priority, the device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority which is placed last in the series?
    Parallel
45. Identify the type of serial communication error condition in which A 0 is received instead of a stop bit (which is always a 1)?
    Framing error
46. Where does the processor store the address of the first instruction of the ISR?
    Interrupt vector
47. In _____ a separate address space of the CPU is reserved for I/O operation
    Isolated I/O
48. Every time you press a key, an interrupts is generated. This is an example of
    All of the given
49. _____ is an electrical pathway through which the processor communication with the internal and external devices attached to the computer.

    Computer Bus

50. ---------- is the time needed by the CPU to recognize (not service) an interrupt request
    Interrupt Latency
51. _____ is a technique in which some of the CPU's address lines forming an input to the address decoder are ignored.
    Partial decoding
52.

AL-JUNAID TECH INSTITUE

CS501-Advance Computer Architecture
FINAL TERM MCQS
Prepared by: JUNAID MALIK

AL-JUNAID TECH INSTITUE

ALL answers are verified if found any mistake then Correct ACCORDINGLY

1. A software routine performed when an interrupt is received by the computer iscalled as_

   a. Interrupt
   b. Interrupt handler
   c. Exception
   d. Trap

2. Which of the following pins of the processor is designated for maskable interrupts?

   a. NMI
   b. MI
   c. INTR
   d. RINT

3. ET = ____

   a. CP x IC x T
   b. CPI x IC x T
   c. CPI / IC x T
   d. CPI x IC/T

4. By which file extension does the FALCON-A assembler loads a FALCON-Aassembly file?

   a. .asmfa
   b. .org
   c. .exe
   d. .src

5. In which one of the following methods, does the CPU poll to identify the interrupting module and branches to an interrupt service routine on detecting aninterrupt?

   a. Daisy chain
   b. Software poll
   c. Multiple interrupt lines
   d. All of given option_

_____signal has output direction with respect to printera.

O<7…0>

b. STROBE#

c. INT#

d. ACKNLG#

6. _____is said to occur when a 0 is received instead of a stop bit

a. Framing error

b. Party error

c. Block error

d. Over-run error

7. A component connected to the system bus and having control of it during aparticular bus cycle is called _____

a. Slave component

b. Master component

c. System bus

d. Buffer component

8. The information about the interrupt vector is given in 8-bit from 0 to 7, which istranslated to bit_____on the data bus

a. 16 to 32

b. 11 to 18

c. 0 to 7

d. 8 to 15

9. An interface that can be used to connect the microcomputer bus to_____iscalled as I/O port

a. Flip flop

b. Memory

c. Peripheral devices

d. Multiplexers

10. _____allows a peripheral to read and write memory without interventionby the CPU

a. Programmed I/O

b. Interrupt driven I/O

c. Direct memory access(DMA)

d. Polling

AL-JUNAID TECH INSTITUE

11. Every interrupt handler has an interrupt return (IRET) instruction, this instruction is an example of_____return
    - **a.** NEAR
    - **b.** **FAR**
    - **c.** SHORT
    - **d.** RELATIVE
12. Which I/O technique will be used by a sound card that may need to access datastored in the computer`s RAM?
    - **a.** Programmed I/O
    - **b.** Interrupt driven I/O
    - **c.** **Direct memory access(DMA)**
    - **d.** Polling
13. What should be the behavior of interrupt during critical section?
    - **a.** **Must remain disable**
    - **b.** Must remain enable
    - **c.** Depends on current situation
    - **d.** Only important interrupts be enable
14. Identify the type of serial communication error condition in which "0" isreceived instead of stop bit(which is always a "1")
    - **a.** **Framing error**
    - **b.** Parity error
    - **c.** Overrun error
    - **d.** Under run error
15. The Pentium does allow the use of some part of its____accumulator registerEAX
    - **a.** 8 bits
    - **b.** 16 bits
    - **c.** **32 bits**
    - **d.** 64 bits

16. _is an electrical pathway through which the processor communicates with the internal and external devices attached to the computer
    - **a.** **Computer bus**
    - **b.** **Hazard**
    - **c.** **Memory**
    - **d.** **Disk**

AL-JUNAID TECH INSTITUE

17. Where does the processor store the address of the first instruction of the ISR?
    a. **Interrupt vector**
    b. Interrupt request
    c. Interrupt handler
    d. All of the given options
18. _____is the time needed by the CPU to recognize (not service) aninterrupt request.
    a. **Interrupt latency**
    b. Response deadline
    c. Timer delay
    d. Throughput
19. At the start of the transfer operation in synchronous communication, the masteractivates the_____signal.
    a. **Read**
    b. Enable
    c. Data
    d. Acknowledge
20. Which is the last instruction of the ISR that is to be executed when the ISR terminates?
    a. **IRET**
    b. IRQ
    c. INT
    d. NMI
21. Which one of the following methods for resolving the priority makes use ofindividual bits of a priority encoder?
    a. Daisy-Chaining Priority
    b. Asynchronous Priority
    c. **Parallel Priority**
    d. Semi-synchronous Priority
22. If a character is not available at the beginning of an interval, an_____is saidto occur.
    a. **Under-run Error**
23. Tri-state buffers are used for removing_____.
    a. Instruction collision
    b. bus collision
    c. Instruction contention
    d. **bus contention**
24. When a particular sector is found, the data is transferred to_____.
    a. RAM

b. I/O module
c. Cache memory
d. Instruction register

25. Identify the following type of serial communication error condition:
"The prior character that was received was not still read by the CPU and is overwritten by a new received character."
a. Framing error
b. Parity error
c. Overrun error
d. Under-run error

26. Taking control of the system bus for a few bus cycles is known as _____.
a. Bus Stealing
b. Cycle Stealing
c. Cycle Transferring
d. None of given

27. The average latency to the desired data is halfway round the disk so, what willbe the average rotation latency of the disk rotates at 20,000rpm.
a. 1.25ms
b. 1.5ms
c. 1.0ms
d. 2.0ms

28. What is the status of the ACKNLG# signal when a character is completelyreceived by the printer?
a. It goes from low to high
b. . It goes from high to low page 239
c. It toggles its state
d. It remains unaffected

29. Interrupt driven I/O is better than_____.
a. Polling
b. Data forwarding
c. Stall
d. First In First Out

31. Select the parts of a hard disk.
a. Header only
b. Data section and a trailer
c. Data section only

d. Header, data section and a trailer

32. In which one of the following methods for resolving the priority, the device with the highest priority is placed in the first position, followed by lower- priority devices up to the device with the lowest priority, which is placed last inthe series?
    - **a.** Asynchronous
    - **b. Daisy-Chaining Priority**
    - **c.** Parallel
    - **d.** Semi-synchronous
33. Identify the following type of serial communication error condition in which nocharacter is available at the beginning of an interval.
    - **a.** Framing error
    - **b.** Parity error
    - **c.** Overrun error
    - **d. Under-run error**
34. In the little-endian format exchanging data between computer, the data transmitted by one will be received in a "swapped" form by the other.
    - **a.** Organized
    - **b.** Signals
    - **c. Swapped**
    - **d.** Arranged
35. The source file of FALSIM should contain_____text only.
    - **a.** Unicode
    - **b. ASCII**
    - c. ANSI
    - d. UTF

36. A component connected to the_____and with which the master componentcan communicate during a particular bus cycle. Normally the CPU with its bus control logic is the master component.
    - **a.** Slave component
    - **b. System bus**
    - **c.** Master component
    - **d.** Bus component

37. In which technique does the hardware directly access host memory for readingor writing

independent of CPU?

a. Direct Memory Access (DMA)
b. Programmed I/O
c. Interrupt driven I/O
d. Polling

38. Most parallel I/O ports used with peripheral devices are mapped on a range of _____.

a. Bus addresses
b. Direct memory access
c. Contiguous addresses
d. Cache

39. _____signal is used in printer with DB-25 interface to reset its controller.

a. #PE
b. #STROB
c. #INIT
d. #SLCT

40. Why DMA is faster than Programmer I/O technique because?

a. DMA transfers data directly using CPU
b. DMA transfers data directly without using CPU CONCEPTUAL
c. DMA uses buffers with CPU
d. DMA uses interrupted driven I/O

41. _____is a technique in which some of the CPU's address lines formingan input to the address decoder are ignored?

- Microprogramming

b. Instruction pre-fetching
c. Pipelining
d. Partial decoding

42. In 8086/8088 processor, interrupt vector table is located at the memory location _____.

a. 0
b. 4
c. 256
d. 1024

43. When an I/O module has a capability of executing a specific set of instructionsfor specific I/O devices in the memory without the involvement of CPU is called _____

a. Selector Channel

b. I/O Channel

c. I/O processors

d. Cycle Stealing

44. How does DMA saves CPU time?

a. By controlling transfer between I/O devices and memory directly

b. By storing all data in a buffer to be later transferred to the CPU

c. By periodically polling

d. By issuing a interrupt request to the CPU to request attention 45.Connection to a CPU that provides a data path between the CPU and external devices, such as a keyboard, display, or reader is called _____

a. Buffer

b. I/O port

c. Memory mapping

d. Processor

46. _____lets the user execute the program, one instruction at a time.

a. Single Step

b. Execute

c. Change PC

d. List File

47. In_____a separate address space of the CPU is reserved for I/O operations.

a. Isolated I/O

b. Memory Mapped I/O

c. All of above

d. None of above

48. Which one of the following is NOT a technique used when the CPU wants toexchange data with a peripheral device?

a. Direct Memory Access (DMA)

b. Interrupt driven I/O

c. Programmed I/O

d. Virtual Memory

49. A computer interface is an_____circuit that matches the requirement of thetwo subsystems between which it is connected.

a. Digital

b. Electronic
c. Primary
d. Obituary

50. _____the device usually means reading its status register every so oftenuntil the device's status changes to indicate that has completed the request.
a. Interrupting
b. Masking
c. Polling
d. Executing

51. For input ports, the incoming data should be placed on the data bus only duringthe I/O read bus cycle. For this purpose,_____are used.
a. Flip Flops
b. Tri-state Buffers
c. AND Gates
d. Registers

52. Which of the following is not true regarding serial communication?
a. Easy to implement
b. Inefficient
c. High cost
d. Slow

53. In a printer with DB-25 interface,_____signal is better for edgetriggered systems.
a. BUSY#
b. PE#
c. ACKNLG#
d. STROB#

54. The_____can be determined from the number of platters and the numberof tracks.
a. Speed of processing
b. Execution time
c. Storage capacity
d. Latency

55. The directive_____is used to define variables.
a. .equ
b. .db

d. .org

56. _____means that the CPU should input data from an input device only when the device is ready to provide data and send data to an output device onlywhen it is ready to receive data.

a. Data location
b. Data synchronization
c. Data transfer
d. Asynchronous transmission

57. The main issue/s in error control is/are_____.

a. Detection of Error
b. Correction of Error
c. Both Detection of Error and Correction of Error
d. Avoidance of Error

58. _____signal has input direction with respect to printer

a. BUSY
b. STROBE#
c. PE#
d. ACKNLG#

59. A parallel port can be considered to be a big_____gate.

a. OR
b. AND
c. NOR
d. NOR

60. Every time you press a key, an interrupt is generated. This is an example of

a. Hardware interrupt
b. Software interrupt
c. All of the given
d. None of the given

61. How Interrupt driven I/O is better than polling because?

a. Interrupt driver I/O is easy to design
b. Interrupt driver I/O is enhanced version of polling
c. Interrupt driver I/O does not waste time on checking which device isavailable
d. Interrupt driven I/O is easy to program62.How can you define an interrupt?

a. A process where an external device can speedup the working of the microprocessor
b. A process where memory can speed up programs execution speed

c. **A process where an external device can get the attention of the microprocessor**
d. A process where input devices can takeover the working of the microprocessor

63. _____is/are example(s) of synchronous communication.
a. Register to Register
b. Register to Memory
c. Memory to Memory
d. All of the given

64. _____depends upon the present position of the head and the position ofthe required sector.
a. Direct memory Access
b. Execution time
c. Throughput
d. Seek time

65. Which one of the following is the memory organization of **SRC processor?**

- 2^8 * 8 bits
- 2^16 * 8 bits
- **2^32 * 8 bits (Page 46)**
- 2^64 * 8 bits

66. Type A format of SRC uses instructions

- **Two (Page 47)**
- three
- four
- five

67. The instruction ---- will **load** the register R3 with the contents of the memory location M [PC+56]
- Add R3, 56
- lar R3, 56
- **ldr R3, 56 (Page 47)**
- str R3, 56

68. Which format of the instruction is called the accumulator?
- 3-address instructions
- 3-address instructions
- 2-address instructions
- **1-address instructions (Page 32)**
- 0-address instructions

69. Which one of the following are the **code size** and the **Number of memory bytes** respectively for a 2-address instruction?
- 4 bytes, 7 bytes

- **7 bytes, 16 bytes (Page 36)**
- 10 bytes, 19 bytes
- 13 bytes, 22 bytes

70. Which operator is used to name registers, or part of registers, in the Register Transfer Language?
- **:= (Page 66)**
- &
- %
- ©

71. The transmission of data in which each character is self-contained units with its own start and stop bits is ------
- **Asynchronous**
- Synchronous
- Parallel
- All of the given options

72. Circuitry that is used to move data is called ----------
- **Bus**
- Port
- Disk
- Memory

73. Which one of the following is **NOT** a technique used when the CPU wants to exchange data with a peripheral device?
- Direct Memory Access (DMA).
- Interrupt driven I/O
- Programmed I/O
- **Virtual Memory (Page 268)**

74. Every time you press a key, an interrupt is generated. **This is an example of**
- **Hardware interrupt (Page 275)**
- Software interrupt
- Exception
- All of the given

75. The interrupts which are pre-programmed and the processor automatically finds the address of the ISR using interrupt vector table are
- Maskable
- Non-maskable
- Non-vectored
- **Vectored (Page 277)**

76. Which is the last instruction of the ISR that is to be executed when the ISR terminates?
- **IRET (Page 278)**
- IRQ
- INT
- NMI

77. If NMI and INTR both interrupts occur simultaneously, then which one has the precedence over the other
- **NMI (Page 279)**
- INTR

- IRET
- All of the given

78. Identify the following type of serial communication error condition:

The prior character that was received was not still read by the CPU and is over written by a new received character.

- Framing error
- Parity error
- **Overrun error (Page 240)**
- Under-run error

79. ----------the device usually means reading its status register every so often until the device's status changes to indicate that it has completed the request.

- Executing
- Interrupting
- Masking
- **Polling**

80. Which I/O technique will be used by a sound card that may need to access data stored in the computer's RAM?

- Programmed I/O
- Interrupt driven I/O
- **Direct memory access(DMA)**
- Polling

81. For increased and better performance we use_which are usually made of glass.

- Coaxial Cables
- Twisted Pair Cables
- **Fiber Optic Cables (Page 390)**
- Shielded Twisted Pair Cables

82. In__if we find some call party busy we can have provision of call waiting.

- **Delay System (Page 381)**
- Loss System
- Single Server Model
- None of the given

83. In_technique memory is divided into segments of variable sizes depending upon the requirements.

- Paging
- **Segmentation (Page 365)**
- Fragmentation
- None of the given

84. For a request of data if the requested data is not present in the cache, it is called a __

- **Cache Miss (Page 358)**
- Spatial Locality
- Temporal Locality
- Cache Hit

85. An entire_memory can be erased in one or a few seconds which is much faster than EPROM.

- PROM
- Cache

- EEPROM
- **Flash Memory (Page 356)**

86. ____chips have quartz windows and by applying ultraviolet light data can be erased from them.
- PROM
- Flash Memory
- **EPROM (Page 356)**
- EEPROM

87. The___signal coming from the CPU tells the memory that some interaction is required between the CPU and memory.
- **REQUEST (Page 350)**
- COMPLETE
- None of the given

88. ___is a combination of arithmetic, logic and shifter unit along with some multiplexers and control unit.
- Barrel Rotator
- Control Unit
- Flip Flop
- **ALU (Page 347)**

89. In Multiple Interrupt Line, a number of interrupt lines are provided between the____modules
- **CPU and the I/O (Page 283)**
- CPU and Memory
- Memory and I/O
- None of the given

90. The data movement instructions____data within the machine and to or from input/output devices.
- Store
- Load
- Move
- **None of given (Page 141)**

91. CRC has----- overhead as compared to Hamming code.
- Equal
- Greater
- **Lesser (Page 329)**
- None of the given

92. The_____is w-bit wide and contains a data word, directly connected to the data bus which is b-bit wide memory address register (MAR) .
- Instruction Register(IR)
- memory address register (MAR)
- **) memory Buffer Register(MBR) (Page 350**
- Program counter (PC)

93. In___technique, a particular block of data from main memory can be placed in only one location into the cache memory .
- Set Associative Mapping
- **Direct Mapping(Page 360)**
- Associative Mapping
- Block Placement

94. ____indicate the availability of page in main memory.
- Access Control Bits
- Used Bits
- **Presence Bits**
- None of the given

95. The_____RTN describes the overall effect of instructions on the programmer visible registers.
- ►**Abstract**
- ►Concrete
- ►Absolute
- ►Basic

96. The instruction set is of_____importance in governing the structure and function of the pipeline.
- ►Least
- ►**Primary**
- ►Secondary
- ►No

97. _____is the most general and least useful performance metrics for RISC machines.
- ►**MIPS**
- Instruction Count
- ►Number of registers
- ►Clock Speed

98. A_____provides four functions: Select, DataIn, DataOut and Read/Write.
- ►ALU
- ►Bus
- ►Register
- ►**Memory Cell (Page 351)**

99. We can classify or partition the SRC instructions by their overall__behavior.
- ►**Register transfer**
- ►Memory transfer
- ►Execution
- ►Logical

100. The__RTN describes detailed register transfer steps in the data path that produce the overall effect.
- ►Abstract
- ►**Concrete**
- ►Absolute
- ►Basic

101. All members of the MC68000 family are_____processors.

►**32-bit**
►16-bit
►64-bit
►8-bit

102. _____Operations refers to a processor that can issue more than one instruction simultaneously.

►Macro
►Micro
►Scalar
►**Superscalar**

103. Exceptions which are___occur in response to events that are paced by the internal processor clock.

► Asynchronous
► **Synchronous**
► Internal
► External

104. In the hazard detection by hardware, resolved by pipeline stalls, if the instructions are in the adjoining stages, then the hazard must be detected in stage___.

► 4
► 2
► **3**
► 1

105. 1-bit sign, 8-bit exponent, 23-bit fraction and a bias of 127 is used for__Binary Floating Point Representation

►Double precision
►**Single Precision (Page 348)**
►All of above
►Half Precision

106. The average rotational latency if the disk rotated at 20,000rpm is ____

►0.5 ms
►3.5 ms
►2.5 ms
►**1.5 ms (Page 324)**

107. A hard disk with 5 platters has 1024 tracks per platter, 512 sectors per track and 512 bytes/sector. What is the total capacity of the disk?

►1.5 GB
►**1 GB (Page 324)**
►2 GB
►3 GB

108. Where does the processor store the address of the first instruction of the ISR?

- **Interrupt vector (Page 277)**
- Interrupt request
- Interrupt handler
- All of the given options

109. In____, a separate address space of the CPU is reserved for I/O operations.

- **Isolated I/O (Page 236)**
- Memory Mapped
- I/O All of above
- None of above

110. ---------- is the time needed by the CPU to recognize (not service) an interrupt request.

- **Interrupt Latency (Page 279)**
- **Response**
- **Deadline Timer**
- **delay Throughput**

111. How can you define an interrupt?

- A process where an external device can speedup the working of the microprocessor
- A process where memory can speed up programs execution speed
- A process where an external device can get the attention of the microprocessor
- A process where input devices can takeover the working of the microprocessor

112. A software routine performed when an interrupt is received by the computer is called as ---------

- Interrupt
- **Interrupt handler**
- Trap

113.
In which one of the following methods for resolving the priority, the device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the series?

- Asynchronous
- **Daisy-Chaining Priority**
- Parallel
- Semi-synchronous

114.
Identify the type of serial communication error condition in which A 0 is received instead of a stop bit (which is always a 1)?

- **Framing error (Page 240)**

- Parity error
- Overrun error
- Under-run error

115. Identify the following type of serial communication error condition in which no character is available at the beginning of an interval.

- Framing error
- Parity error
- Overrun error
- Under-run error (Page 240)

116.

A---------- is a wiring scheme in which, for example, device A is wired to device B, device B is wired to device C, device C is wired to device D etc.

- **Daisy chain**
- DMA
- Interrupt driven I/O
- Polling

117.

An --------- is the memory address of an interrupt handler.

- **Interrupt vector**
- Interrupt service *
- routine Exception
- Mask

118.

The conversion of numbers from a representation in one base to another is known as___

- **Radix Conversion (Page 333)**
- Number Representation
- Decimal representation
- Hexadecimal Representation

119.

In which one of the following interrupts the device have to supply the address of the subroutine to

the Microprocessor

- Maskable
- **Non-maskable**
- Non-vectored
- Vectored

120.

---------- interrupts are usually associated with the

- software hardware
- **software**
- machine
- internal

121.

How Interrupt driven I/O is better than polling because?

- Interrupt driver I/O is easy to design
- Interrupt driver I/O is enhanced version of polling.
- **Interrupt driver I/O does not waste time on checking which device is available. (Page 274)**
- Interrupt driven I/O is easy to program.

122.

In Single-Precision Binary Floating Point Representation the exponent is _

- **8 bits (Page 348)**
- 11 bits
- 1 bit
- 23 bits

123.

The_____is m-bits wide and contains memory address generated by the CPU directly connected to the m-bit wide address bus Booth Recording

- **memory address register (MAR) (Page 350)**
- memory Buffer Register (MBR)
- Program counter (PC)
- InstructionRegister(IR)

124.

A combination of parallel and sequential hardware used to build a multiplier is known as _

- Parallel Array
- Multiplier Booth Recording
- **Series Parallel Multiplier (Page 342)**
- None of the given

AL-JUNAID TECH INSTITUE

125.
The register file is a collection of_bit wide registers used for data transfer between memory and the CPU .

- 8
- 16
- 32 (Page 350)
- 64

126.
The_____of an m digit number x is xc'=bm -1- x

- Radix Compliment
- **Diminished Radix Compliment (Page 337)**
- Signed Magnitude Form
- Biased Representation

127.
Shifting of the radix point towards left or right

- Shifting
- Logical
- Shift Right Shift
- Scaling (Page 335)

128.
In_____adder circuit we feed carry out from the previous stage to the next stage and so on.

- **Ripple Carry Adder (Page 341)**
- Carry Look Ahead Adder
- Complement Adder
- 2's Complement Adder

129.
_____are computed by the ALU and stored in processor status register.

- **Condition codes(Page 334)**
- Conditional Branches Fraction
- Division
- None of the

given

130.
A_____signal decides whether the input word should be shifted or bypassed.

- Control Read
- **Shift/bypass (Page 346)**
- Control Write
- None of the given

131.
In____recording ,bits are encoded in pairs so there are only ' n/2' additions instead of 'n'.

- Booth Recording
- **Bit Pair Recording (Page 343)**
- Integer division
- None of the given

132.
Given an m-digit base b number x, the_of x is xc = (bm– x) mod bm

- **Radix Compliment (Page 337)**
- Diminished Radix
- Compliment Signed
- Magnitude Form

Biased Representation

133.
For_____of an error we just need to know that there exists an error.

- **Detection (Page 328)**
- Correction
- Both Correction and Detection
- None of the give

134.
In Double-Precision Binary Floating Representation the function is_

- 23 bits
- **52 bits (Page 348)**
- 1 bits

- 1 bit

135.

_____is the simplest form for representing a signed number

- Based representation
- Diminshed Redex Complement Form
- **Sign Magnitude Form (Page 336)**
- None of the given

136.

In computers, floating-point representation uses__to encode significand, exponent and their sign in a single word

- Decimal Numbers
- **Binary Numbers (Page 347)**
- Octal Numbers
- Hexa decimal Numbers

137.

Which one of the following registers store a previously calculated value or a value loaded from the main memory?

► **Accumulator**
► Address Mask
► Instruction Register
► Program Counter

138.

Which one of the following portions of an instruction represents the operation to be performed?

► Address
► Instruction code
► **Opcode (Page 33)**
► Operand

139.

_____control signal enable the input to the PC for receiving a value that is currently on the internal processor bus.

► **LPC (Page 172)**
► INC4
► LC
► Cout

140.

What is the instruction length of the FALCON-E processor?

- 8 bits
- 16 bits
- **32 bits (Page 134)**
- 64 bits

141.

Which type of instructions enables mathematical computations?

► **Arithmetic (Page 92)**
► Control
► Data transfer
► None of the given

142.
What is the instruction length of the SRC and Falcon E processor?
► 8 bits
► 16 bits
► **32 bits (Page 134)**
► 64 bits

143. An instruction that specifies one operand in memory and one operand in a register would be known as a _____ address instruction.
►2-1/2
► **1-1/2 (Page 37)**
►0
►2

144.
In floating point representations__is also called mantissa.
► Sign
► Base
► **Significant (Page 347)**
► Exponent

145. What should be the behavior of interrupts during critical sections?
► **Must remain disable (Page 197)**
► Must remain Enable
► Can be either enable or disable
► only important interrupts be enable

148. Which one of the following is a binary cell capable of storing one bit of information?
► Decoder
► **Flip-flop (Page 76)**
► Multiplexer
► Diplexer

149. Which type of instructions load data from memory into registers, or store data from register into memory and transfer data between different kinds of special-purpose registers?
► Arithmetic
► Control
► **Data transfer (Page 88)**
► Floating point

150.
What does the RTL expression [M(1234)] means?
► **The contents of memory whose address is 1234.**
► The contents of data register 1234
► The effective address of register 1234
► The address of memory whose address is 1234.

151. Which one of the following languages presents a simple, human-oriented language to specify the operations, register communication and timing of the steps that take place within a CPU to carry out higher level (user programmable) instructions?

► Assembly Language
► OOP(Object Oriented Language)
► **RTL (Register Transfer Language)**
► UML(Unified Modeling language)

152.
Which one of the following instructions is used to load register from memory using a relative address?
► la
► lar

► **ldr (Page 145)**
► str

153.
Taking control of the system bus for a few bus cycles is known as____.

► Bus Stealing
► **Cycle Stealing (Page 317)**
► Cycle Transfering
► None of given

154. In-----address mode, the actual data is stored in the instruction.

► Direct
► Indirect
► **Immediate**
► Relative

155. Keyboard Interrupt (INT 9) is an example of interrupt.

► **Hardware**
► Software

156. A user program has to delete a file. The user program will be executing in the user mode. When it makes the specific system call to delete the file, an interrupt will be generated, this will cause the processor to halt its current activity and switch to supervisor mode. Once in supervisor mode, the operating system will delete the file and then control will return to the user program. This is an example of

► Hardware interrupt
► **Software interrupt (Page 275)**
► Exception
► All of the given

157. By which file extension does the FALCON-A Assembler loads a FALCON-A assembly file?

► **.asmla (Page 8)**

►.org
►.exe
►.src

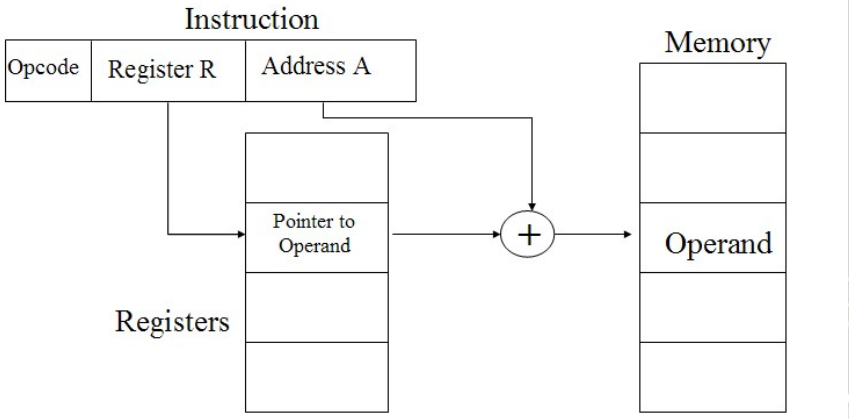
157.
All ----------interrupts have priority over all interrupts

►**internal, external (Page 279)**
►external, internal

158. The can also be used anywhere in the source file to force code at a particular address in the memory.

►.end directive
►.start directive
►.label directive
►**.org directive (Page 298)**

Question No:106



In this figure, the constant value specified by the immediate field is added to the register value, and the resultant is the index of memory location that is referred i.e. Effective Address = A + (content of R) .
Identify the addressing mode.

►**Displacement(Page 139)**
►Immediate
►Indexed
►Relative

159. In which one of the following addressing modes, the operand does not specify an address but it is the actual data to be used.

►Direct

- Indirect
- **Immediate**
- Relative

160. When is the “Divide error interrupt‖ generated?

► When an attempt is made to divide by decimal number
► When an attempt is made to multiply by zero
► **When an attempt is made to divide by zero (Page 197)**
► When negative number is stored in a register

161. Which one of the following is a term used to describe a storage systems' resilience to disk failure through the use of multiple disks and by the use of data distribution and correction techniques?

► Interrupt handling
► Programmed I/O
► Polling
► **RAID** click here for detail

162. _____is the time for first bit of the message to arrive at the receiver including delays.
► Transmission Time
► Latency
► Transport Latency
► **Time of Flight (Page 388)**

163.
Falcon-A Simulator loads a FALCON-A binary file with a_extension and presents its contents into different areas of the simulator.

► .bin
► **.binfa (Page 5)**
► .fa
► None of the given

164. In machines where instructions can be executed in parallel or out of order, two additional hazards can occur: WAW and ----------

► None fo the given
► **WAR**
► RAW
► RAR

165. For___of an error we just need to know that there exists an error.

► None of the given
► Correction
► **Detection (Page 328)**
► Both Correction and Detection

166. Identify the type of serial communication error condition in which 0 is

received instead of a stop bit (which is always a 1)?

► **Framing error (Page 240)**
► Parity error
► Overrun error
► Under-run error

167._____is/are defined as the number of instructions processed per second

► **Throughput (Page 203)**
► Latency Time to process 1 request.
► Throughput and Latency
► None of the given

168. Raid Level_is not a true member of the RAID family.

► **0 (Page 330)**
►2
►3
►4

169. Which one of the following is an address (binary bit pattern) issued by CPU?

►Memory
► **Effective (Page 39)**
►Base
►Next instruction

170. Which one the following interrupts is initiated with an INT instruction?

►Hardware
► **Software**
►Both hardware and Software
►None of the given

171. An -- is a program that takes basic computer instructions and converts them into a pattern of bits that the
computer's processor can use to perform its basic operations.

► **Assembler**
►Debugger
►Editor
►Console

172. Dirty bit is a status bit which is used to indicate whether_____.

a. The block is accessible or not
b. The block has been modified or not page 327
c. The block is valid or not

d. The block has been accessed frequently or not

173. In 1x8 memory cell arrangement, each block is connected through a bi-directional data bus implemented with___tri-state buffer(s).

a. 1

b. 2 page 317

c. 4

d. 8

174. The register file is a collection of___bit wide registers used for data transfer between memory and the CPU.

a. 8

b. 16

c. 32 page 316

d. 64

175. Human works with base 10 and computers work with base_____.

a. 8

b. 10

c. 2 page 301

c. 16

176. Raid level___distributes the parity strips across all disks.

a. 2

b. 3

c. 4

d. 5 page 300

177. Shifting of the radix point towards left or right is called _____

a. Shifting

b. Logical Shift

c. Right Shift

d. Scaling page 302

178. In computers, floating-point representation uses_____to encode significand, exponent and their sign in a single word
    a. Decimal Numbers
    b. Binary Numbers page 313
    c. Octal Number
    d. Hexa decimal Numbers
179. The_____is w-bit wide and contains a data word, directly connected to the data bus which is b-bit wide memory address register (MAR).
    a. Memory Buffer Register (MBR) page 316
    b. Program Counter (PC)
    c. Instruction Register (IR)
    d. Memory Address Register (MAR)
180. Adding a data pin to a chip with 2^m words of s bits increases the number ofbits it can store by only a factor of ___
    a. s/(s+1)
    b. (s+1)/s page 320
    c. (s+2)/s
    d. s^2/s
181. A given block in cache is identified uniquely by its main memory blocknumber, referred to as___.
        i. Ticket
        ii. Serial
    c. Tag page 323
    d. Label
182. The conversion of numbers from a representation in one base to another isknown as___.
    a. Radix Conversion page 301
    b. Number Representation
    c. Decimal representation
    d. Hexadecimal Representation
183. Multiple copies of the same data can exist in memory hierarchy simultaneously. The Cache needs updating mechanism to prevent old datavalues from being used. This is the problem of____.
    a. Cache Miss
    b. Dirty bit
    c. Cache Coherence page 327
    d. Write Allocate

184. Raid Level_____is not a true member of the RAID family.

a. 0 page 298

b. 2

c. 3

d. 4

185. In Double-Precision Binary Floating Point Representation the fraction is

_____

a. 23 bits

b. 52 bits page 314

c. 11 bits

d. 1 bit

186. ___ is nonvolatile and may be written into only once.

a. PROM page 321

b. EPROM

c. EEPROM

d. Main memory

187._____ is non volatile i-e it retains the information in it when power is removed from it

a. RAM

b. Hard Disc

c. ROM page 320

d. Cache

188. A typical one level decoder has___input(s) and_____ output(s).

i. n, n

ii. 2^n, n

iii. n, n^2

d. n, 2^n page 318

189. Along with the information bits, we add up another bit, which is called?

a. Start bit

b. Header bit

c. Parity bit page 297

d. Stop bit

190. Which of the following is NOT a function of memory cell?

a. Activate page 317

b. DataIn

c. DataOut

d. Read/Write

191. The____of an m digit number x is \[{x^c} = {b^m} - 1 - x\]
   a. Radix Compliment
   b. Diminished Radix Compliment page 304
   c. Signed Magnitude Form
   d. Biased Representation

192. The memory management unit (MMU) is located between_____and _____.
   a. Main memory and secondary memory
   b. The CPU and the physical memory page 328
   c. Secondary memory and Virtual memory
   d. ROM and RAM

193. Given an m-digit base b number x, the_____of x is

$$x^c = (b^m - x) \bmod b^m$$

   a. Radix Compliment page 304
   b. Diminished Radix Compliment
   c. Signed Magnitude Form
   d. Biased Representation

194. For___ of an error we just need to know that there exists an error.
   a. Detection page 297
   b. Correction
   c. Both Correction and Detection
   d. None of the given

195. __ is much faster than EPROM.
   a. Main memory
   b. Rom
   c. Hard disk
   d. Flash Memory page 321

196. CRC has_____overhead as compared to Hamming code.
   a. Equal
   b. Greater
   c. Lesser page 298
   d. None of the given

197. very large page size results in increased_____.
   a. Through put
   b. access time page 330
   c. Delay
   d. Execution time

198. For write to complete in Write through, the CPU has to wait. This wait stateis called_.
   a. Write Buffer
   b. Cache Miss
   c. Write Allocate
   d. Write Stalls page 327

199. The___signal coming from the CPU tells the memory that some interaction is required between the CPU and memory.
   a. REQUEST page 316
   b. R/W
   c. COMPLETE
   d. None of the given

200. A 64kx1 Static RAM Chip has a cell array which consists of _____ row(s) and_____column(s).
   a. 64, 1
   b. 1, 64
   c. 64, 256
   d. 256, 256 page 317

201. _____ chips have quartz windows and by applying ultraviolet light data can be erased from them.
   a. PROM
   b. Flash Memory
   c. EPROM page 321
   d. EEPROM

202. ___ is the concept in which a process is copied into the main memory from the secondary memory according to the requirement.
   a. Paging
   b. Demand Paging page 329
   c. Segmentation
   d. Logical Partition

203. In virtual memory mechanism, pages are formulated in the _____ memory and brought into the_____memory.
   a. Secondary, cache
   b. Main, cache
   c. Main, secondary
   d. Secondary, main page 328

204. Which is a status bit that indicates whether the block in cache has been modified or not modified?
    a. Presence bit
    b. Dirty bit page 327
    c. Access bit
    d. End bit
205. _____ refers to the fact that once a particular data item is accessed, it is likely that it will be referenced again within a short period of time.
    a. Spatial Locality
    b. Temporal Locality page 322
    c. Full Locality
    d. Half Locality
206. combination of parallel and sequential hardware used to build a multiplieris known as _____
        i. Parallel Array Multiplier
        ii. Both Recording
        iii. Series Parallel Multiplier
        iv. None of the given
207. When_____signal is high, this would correspond to a read operation equivalent to having an input data to the CPU and output from the memory.
    a. R/W page 316
    b. COMPLETE
    c. REQUEST
    d. None of the given
208. The_____ is m-bits wide and contains memory address generated by the CPU directly connected to the m-bit wide address bus Booth Recording
    a. memory address register (MAR) page 316
    b. memory Buffer Register(MBR)
    c. Program counter (PC)
    d. Instruction Register(IR)
209. Adding an address pin to a memory chip increases the capacity of memoryby a factor of__.
    a. 1.5
    b. 2 page 320
    c. 2.5
    d. 3

210. ___ is a read-mostly memory that can be written into at any timewithout erasing prior contents

a. PROM
b. EPROM
c. Flash Memory
d. EEPROM page 321

211. A 16kx4 Static RAM Chip is arranged in the form of four_____memory cells

a. 64x256 page 318
b. 16x4
c. 4x16
d. 256x256

212. _____ is the simplest form for representing a signed number

a. Biased Representation
b. Diminished Radix Compliment Form
c. Sign Magnitude Form page 304
d. None of the given

213. The Direct memory access (DMA) scheme results in direct link between _____ and_____.

a. the CPU and the physical memory
b. main memory and secondary memory page 331
c. Secondary memory and Virtual memory
d. Cache memory and Registers

214. An entire_____ memory can be erased in one or a few seconds whichis much faster than EPROM.

a. PROM
b. Cache
c. EEPROM
d. Flash page 321

215. _____ refers to the fact when a given address has been referenced, the next address is highly probable to be accessed within a short period of time

a. Temporal Locality
b. Spatial Locality page 322
c. Full Locality
d. Half Locality

216. A _____signal decides whether the input word should be shifted orbypassed.

a. Control Read

b. shift/bypass page 312

c. Control Write

d. None of the given

217. In___ adder circuit we feed carry out from the previous stage to the next stage and so on.

a. Ripple Carry Adder page 308

b. Carry Look Ahead Adder

c. Complement Adder

d. 2's Complement Adder

218. In Single-Precision Binary Floating Point Representation the exponent is _____

a. 8 bits page 313

b. 11 bits

c. 1 bit

d. 23 bits

219. Each memory reference issued by the CPU is translated from the logicaladdress space to_____.

a. Effective address

b. Physical address page 328

c. Virtual address

d. Cache address

220. _____ are computed by the ALU and stored in processor status register.

a. Condition codes page 311

b. Conditional Branches

c. Fraction Division

d. None of the given

221. In_____, bits are encoded in pairs so there are only 'n/2' additions instead of 'n'.

a. Booth Recording

b. Bit Pair Recording page 309

c. Integer division

d. None of the given

222. __ is a combination of arithmetic, logic and shifter unit along withsome multiplexers and control unit.

a. Barrel Rotator
b. Control Unit
c. Flip Flop
d. ALU page 313

223. The cache contains a copy of portions of the_____.

a. Main memory page 321
b. Rom
c. EPROM
d. Flash memory

224. What is the basic idea of "carry look ahead"?

a. To reduce congestion
b. To speed up the ripple carry page 308
c. To solve the redundancy
d. To synchronize with CPU clock

225. Along with the information bits we add up another bit which is called the _____ bit.

a. CRC
b. Hamming
c. Error Detection
d. Parity page 297

226. Virtual memory acts as a cache between_____and_____.

a. Secondary memory and Virtual memory
b. Cache memory and Registers
c. ROM and RAM
d. Main memory and secondary memory page 328

227. - Please choose one Which one of the following is the memory organization of SRC processor?

- 2^8 * 8 bits
- 2^16 * 8 bits
- 2^32 * 8 bits (Page 46)
- 2^64 * 8 bits

228. Please choose one Type A format of SRC uses ----------instructions

- Two (Page 47)
- three
- four
- five

229. - Please choose one The instruction ----------will load the register R3 with the contents of the memory location M [PC+56]

- Add R3, 56

- lar R3, 56
- ldr R3, 56 (Page 47)
- str R3, 56

230. - Please choose one Which format of the instruction is called the accumulator?
- 3-address instructions
- 2-address instructions
- 1-address instructions (Page 32)
- 0-address instructions

231. Please choose one Which one of the following are the code size and the Number of memory bytes respectively fo a 2-address instruction?
- 4 bytes, 7 bytes
- 7 bytes, 16 bytes (Page 36)
- 10 bytes, 19 bytes
- 13 bytes, 22 bytes

232. - Please choose one Which operator is used to name registers, or part of registers, in the Register Transfer Language?
- := (Page 66)
- &
- %
- ©

233. - Please choose one The transmission of data in which each character is self-contained units with its own start an stop bits is ------
- Asynchronous
- Synchronous
- Parallel
- All of the given options

234. - Please choose one Circuitry that is used to move data is called ----------
- Bus
- Port
- Disk
- Memory

235. - Please choose one Which one of the following is NOT a technique used when the CPU wants to exchange data with a peripheral device?
- Direct Memory Access (DMA).
- Interrupt driven I/O
- Programmed I/O
- Virtual Memory (Page 268)

236. Please choose one Every time you press a key, an interrupt is generated. This is an example of
- Hardware interrupt (Page 275)
- Software interrupt
- Exception
- All of the given

237. - Please choose one The interrupts which are pre-programmed and the processor automatically finds the address o the ISR using interrupt vector table are
- Maskable
- Non-maskable
- Non-vectored
- Vectored (Page 277)

238. - Please choose one Which is the last instruction of the ISR that is to be executed when the ISR terminates?
- IRET (Page 278)

- IRQ
- INT
- NMI

239. - Please choose one If NMI and INTR both interrupts occur simultaneously, then which one has the precedence over the other
- NMI (Page 279)
- INTR
- IRET
- All of the given

240. Identify the following type of serial communication error condition: The prior character that was received was no still read by the CPU and is over written by a new received character.
- Framing error
- Parity error
- Overrun error (Page 240)
- Under-run error

241. ---------the device usually means reading its status register every so often until the device's status changes to indicate that it has completed the request.
- Executing
- Interrupting
- Masking
- Polling

242. - Please choose one Which I/O technique will be used by a sound card that may need to access data stored in the computer's RAM?
- Programmed I/O
- Interrupt driven I/O
- Direct memory access(DMA)
- Polling

243. - Please choose one For increased and better performance we use _____ which are usually made of glass.
- Coaxial Cables
- Twisted Pair Cables
- Fiber Optic Cables (Page 390)
- Shielded Twisted Pair Cables

244. - Please choose one In _____ if we find some call party busy we can have provision of call waiting.
- Delay System (Page 381)
- Loss System
- Single Server Model
- None of the given

245. In ____ technique memory is divided into segments of variable sizes depending upon the requirements.
- Paging
- Segmentation (Page 365)
- Fragmentation
- None of the given

246. - Please choose one For a request of data if the requested data is not present in the cache, it is called a _____
- Cache Miss (Page 358)
- Spatial Locality
- Temporal Locality
- Cache Hit

247. Please choose one An entire _____ memory can be erased in one or a few seconds which is much faster than EPROM.

- PROM
- Cache
- EEPROM
- Flash Memory (Page 356)

248. - Please choose one _____chips have quartz windows and by applying ultraviolet light data can be erased from them.
- PROM
- Flash Memory
- EPROM (Page 356)
- EEPROM

249. - Please choose one The _____signal coming from the CPU tells the memory that some interaction is required between the CPU and memory.
- REQUEST (Page 350)
- COMPLETE
- None of the given

250. _____ is a combination of arithmetic, logic and shifter unit along with some multiplexers and control unit.
- Barrel Rotator]
- Control Unit
- Flip Flop
- ALU (Page 347)

251. - Please choose one In Multiple Interrupt Line, a number of interrupt lines are provided between the _____ modules.
- CPU and the I/O (Page 283)
- CPU and Memory
- Memory and I/O
- None of the given

252. - Please choose one The data movement instructions _____ data within the machine and to or from input/output devices.
- Store
- Load
- Move
- None of given (Page 141)

253. - Please choose one CRC has ---------- overhead as compared to Hamming code.
- Equal
- Greater
- Lesser (Page 329)
- None of the given

254. - Please choose one The _____ is w-bit wide and contains a data word, directly connected to the data bus which is b-bit wide memory address register (MAR) .
- Instruction Register(IR)
- memory address register (MAR)
- memory Buffer Register(MBR) (Page 350)
- Program counter (PC)

255. In_____technique, a particular block of data from main memory can be placed in only one location into the cache memory .
- Set Associative Mapping
- Direct Mapping (Page 360)
- Associative Mapping
- Block Placement

AL-JUNAID TECH INSTITUE

256. - Please choose one _____ indicate the availability of page in main memory.
- Access Control Bits
- Used Bits
- Presence Bits
- None of the given

257. The _____ RTN describes the overall effect of instructions on the programmer visible registers.
- Abstract
- Concrete
- Absolute
- Basic

258. - Please choose one The instruction set is of _____ importance in governing the structure and function of the pipeline.
- Least
- Primary
- Secondary
- No
- Uestion

259. - Please choose one _____ is the most general and least useful performance metrics for RISC machines.
- MIPS
- Instruction Count
- Number of registers
- Clock Speed

260. - Please choose one A _____ provides four functions: Select, DataIn, DataOut and Read/Write.
- ALU
- Bus
- Register
- Memory Cell (Page 351)

261. Question No: 5 (Marks: 1) - Please choose one We can classify or partition the SRC instructions by their overal _____ behavior.
- Register transfer
- Memory transfer
- Execution
- Logical

262. - Please choose one The _____ RTN describes detailed register transfer steps in the data path that produce the overall effect.
- Abstract
- Concrete
- Absolute
- Basic

263. Please choose one All members of the MC68000 family are _____ processors.
- 32-bit
- 16-bit
- 64-bit
- 8-bit

264. - Please choose one _____ Operations refers to a processor that can issue more than one instruction simultaneously.
- Macro
- Micro

- Scalar
- Superscalar click here for detail

265. - Please choose one Exceptions which are _____ occur in response to events that are paced by the internal processor clock.
- Asynchronous
- Synchronous click here for detail
- Internal
- External

266. - Please choose one In the hazard detection by hardware, resolved by pipeline stalls, if the instructions are in the adjoining stages, then the hazard must be detected in stage _____.
- 4
- 2
- 3
- 1

267. Please choose one 16k x4 static RAM Chip is arranged in the form of four _____ cells.
- 16x512
- 32x512
- 256x512
- 64x256 (Page 352)

268. - Please choose one In a DRAM cell, the storage capacitor will discharge in around _____
- 4 -15 ms (Page 354)
- 2 - 10 ms
- 5-20 ms
- 10-25 ms

269. Please choose one 1-bit sign, 8-bit exponent, 23-bit fraction and a bias of 127 is used for _____ Binary Floating Point Representation
- Double precision
- Single Precision (Page 348)
- All of above
- Half Precision

270. - Please choose one The average rotational latency if the disk rotated at 20,000rpm is _____
- 0.5 ms
- 3.5 ms
- 2.5 ms
- 1.5 ms (Page 324)

271. Question No: 5 (Marks: 3) - Please choose one A hard disk with 5 platters has 1024 tracks per platter, 512 sectors per track and 512 bytes/sector. What is the total capacity of the disk?
- 1.5 GB
- 1 GB (Page 324)
- 2 GB
- 3 GB

272. Where does the processor store the address of the first instruction of the ISR?
- Interrupt vector (Page 277)
- Interrupt request
- Interrupt handler
- All of the given options

273. In _____, a separate address space of the CPU is reserved for I/O operations.
- Isolated I/O (Page 236)

- Memory Mapped I/O
- All of above
- None of above

274. ---------- is the time needed by the CPU to recognize (not service) an interrupt request.
- Interrupt Latency (Page 279)
- Response Deadline
- Timer delay
- Throughput

275. _____ is a technique in which some of the CPU's address lines forming an input to the address decoder are ignored.
- Microprogramming
- Instruction pre-fetching
- Pipelining
- Partial decoding (Page 255)

276. How can you define an interrupt?
- A process where an external device can speedup the working of the microprocessor
- A process where memory can speed up programs execution speed
- A process where an external device can get the attention of the microprocessor
- A process where input devices can takeover the working of the microprocessor

277. An interface that can be used to connect the microcomputer bus to _____is called an I/O Port.
- Flip Flops
- Memory
- Peripheral devices (Page 234)
- Multiplexers

278. A software routine performed when an interrupt is received by the computer is called as ---------
- Interrupt
- Interrupt handler
- Exception
- Trap

279. Which one of the following methods for resolving the priority makes use of individual bits of a priority encoder?
- Daisy-Chaining Priority
- Asynchronous
- Priority Parallel Priority (Page 281)
- Semi-synchronous Priority

280. In which one of the following methods for resolving the priority, the device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the series?
- Asynchronous
- Daisy-Chaining Priority
- Parallel
- Semi-synchronous

281. Identify the type of serial communication error condition in which A 0 is received instead of a stop bit (which is always a 1)?
- Framing error (Page 240)
- Parity error
- Overrun error
- Under-run error

282. Identify the following type of serial communication error condition in which no character is available at the

beginning of an interval.

- Framing error
- Parity error
- Overrun error
- Under-run error (Page 240)

283. _____ is an electrical pathway through which the processor communicates with the internal and external devices attached to the computer.

- Computer
- Hazard
- Memory
- Disk

284. Connection to a CPU that provides a data path between the CPU and external devices, such as a keyboard, display, or reader is called----------

- Processer
- Program
- Buses
- memory address

285. VLIW stands for ----------

- Very Lengthy Interaction Word
- Very Length Instruction Width
- Very Long Instruction Word (Page 219
-) none of given options

286. A ----------is a wiring scheme in which, for example, device A is wired to device B, device B is wired to device C, device C is wired to device D etc.

- Daisy chain
- DMA
- Interrupt driven
- I/O Polling

287. Question # 8 of 10 (Total Marks: 1) Select correct option: An ---------- is the memory address of an interrupt handler.

- Interrupt vector
- Interrupt service routine
- Exception
- Mask

288. The conversion of numbers from a representation in one base to another is known as_____

- Radix Conversion (Page 333)
- Number Representation
- Decimal representation
- Hexadecimal Representation

289. If an interrupt is set by the timer component or by the peripheral device then how would you categorize it?

- Hardware
- Software
- Exception
- All of the given options

290. : 1 In which one of the following interrupts the device have to supply the address of the subroutine to the

- Microprocessor
- Maskable
- Non-maskable click here for detail
- Non-vectored Vectored

291. ---------- interrupts are usually associated with the software

- Hardware
- software
- Machine
- internal

292. When the address of the subroutine is already known to the Microprocessor then it is called as --------- interrupt.
- Maskable
- Non-maskable
- Non-vectored
- Vectored

293. How Interrupt driven I/O is better than polling because?
- Interrupt driver I/O is easy to design
- Interrupt driver I/O is enhanced version of polling.
- Interrupt driver I/O does not waste time on checking which device is available. (Page 274)
- Interrupt driven I/O is easy to program.

294. In Single-Precision Binary Floating Point Representation the exponent is _____
- 8 bits (Page 348)
- 11 bits
- 1 bit
- 23 bits

295. : The_____is m-bits wide and contains memory address generated by the CPU directly connected to the m-bit wide address bus Booth Recording
- memory address register (MAR) (Page 350)
- memory Buffer Register(MBR)
- Program counter (PC)
- Instruction Register(IR)

296. A combination of parallel and sequential hardware used to build a multiplier is known as _____
- Parallel Array Multiplier
- Booth Recording
- Series Parallel Multiplier (Page 342)
- None of the given

297. The register file is a collection of _____bit wide registers used for data transfer between memory and the CPU .
- 8
- 16
- 32 (Page 350)
- 64

298. The _____ of an m digit number x is xc'=bm -1- x
- Radix Compliment
- Diminished Radix Compliment (Page 337)
- Signed Magnitude Form
- Biased Representation

299. Shifting of the radix point towards left or right is called _____
- Shifting
- Logical Shift
- Right Shift
- Scaling (Page 335)

300. Question # 7 of 10 (Total Marks: 1) Select correct option: In _____ adder circuit we feed carry out from the previous stage to the next stage and so on.
- Ripple Carry Adder (Page 341)
- Carry Look Ahead Adder
- Complement Adder

- 2's Complement Adder

301. _____ are computed by the ALU and stored in processor status register.
- Condition codes (Page 334)
- Conditional Branches
- Fraction Division
- None of the given

302. A _____signal decides whether the input word should be shifted or bypassed
- . Control Read
- Shift/bypass (Page 346)
- Control Write
- None of the given

303. Along with information bits we add up another bit which is called the_____bit.
- CRC
- Hamming
- Error Detection
- Parity (Page 328

304. In ____ recording ,bits are encoded in pairs so there are only ' n/2' additions instead of 'n'.
- Booth Recording
- Bit Pair Recording (Page 343)
- Integer division
- None of the given

305. ____ signal is high, this would correspond to a read operation equivalent to having an input data to the CPU and output from the memory REQUEST .
- R/W (Page 350)
- COMPLETE
- REQUEST
- None of the given

306. Given an m-digit base b number x, the _____ of x is xc = (bm– x) mod bm
- Radix Compliment (Page 337)
- Diminished Radix Compliment
- Signed Magnitude Form
- Biased Representation

307. For _____ of an error we just need to know that there exists an error.
- Detection (Page 328)
- Correction
- Both Correction and Detection
- None of the given

308. In Double-Precision Binary Floating Representation the function is____
- 23 bits
- 52 bits (Page 348)
- 1 bits
- 1 bit

309. : _____ is the simplest form for representing a signed number
- Based representation
- Diminshed Redex Complement Form
- Sign Magnitude Form (Page 336)
- None of the given

310. In computers, floating-point representation uses_____ to encode significand, exponent and their sign in a single word
- Decimal Numbers

- Binary Numbers (Page 347)
- Octal Numbers
- Hexa decimal Numbers

311. : Which one of the following registers store a previously calculated value or a value loaded from the main memory?
► Accumulator
► Address Mask
► Instruction Register
► Program Counter

312. Which one of the following portions of an instruction represents the operation to be performed?
► Address
► Instruction code
► Opcode (Page 33)
► Operand

313. _____ control signal enable the input to the PC for receiving a value that is currently on the internal processor bus.
►LPC (Page 172)
► INC4
► LC
► Cout

314. What is the instruction length of the FALCON-E processor?
► 8 bits
► 16 bits
►32 bits (Page 134)
► 64 bit

315. Which type of instructions enables mathematical computations?
►Arithmetic (Page 92)
►Control
►Data transfer
►None of the given

316. What is the instruction length of the SRC and Falcon E processor?
► 8 bits
► 16 bits
► 32 bits (Page 134)
► 64 bits

317. : An instruction that specifies one operand in memory and one operand in a register would be known as a _____ address instruction.
►2-1/2
►1-1/2 (Page 37)
►0
►2

318. In floating point representations _____ is also called mantissa.
►Sign
►Base
►Significant (Page 347)
►Exponent

319. What should be the behavior of interrupts during critical sections?
►Must remain disable (Page 197)
►Must remain Enable
►Can be either enable or disable

►only important interrupts be enable

320. Which one of the following is a binary cell capable of storing one bit of information?
►Decoder
►Flip-flop (Page 76)
►Multiplexer
►Diplexer

321. Which type of instructions load data from memory into registers, or store data from registers into memory and transfer data between different kinds of special-purpose registers?
►Arithmetic
►Control
►Data transfer (Page 88)
►Floating point

322. What does the RTL expression [M(1234)] means?
►The contents of memory whose address is 1234.
►The contents of data register 1234
►The effective address of register 1234
►The address of memory whose address is 1234.

323. Which one of the following languages presents a simple, human-oriented language to specify the operations, register communication and timing of the steps that take place within a CPU to carry out higher level (user programmable) instructions?
►Assembly Language
►OOP(Object Oriented Language)
►RTL (Register Transfer Language)
►UML(Unified Modeling language)

324. Which one of the following instructions is used to load register from memory using a relative address?
►la
►lar
►ldr (Page 145)
►str 23

325. Taking control of the system bus for a few bus cycles is known as _____.
►Bus Stealing
►Cycle Stealing (Page 317)
►Cycle Transfering
►None of given

326. : In----------address mode, the actual data is stored in the instruction.
►Direct
►Indirect
►Immediate
►Relative

327. Keyboard Interrupt (INT 9) is an example of ----- interrupt.
►Hardware
►Software

328. user program has to delete a file. The user program will be executing in the user mode. When it makes the specific system call to delete the file, an interrupt will be generated, this will cause the processor to halt its current activity and switch to supervisor mode. Once in supervisor mode, the operating system will delete the file and then control will return to the user program. This is an example of
►Hardware interrupt
►Software interrupt (Page 275)
►Exception

►All of the given

329. By which file extension does the FALCON-A Assembler loads a FALCON-A assembly file?

►.asmfa (Page 8)

►.org

►.exe

►.src 24

330. All ----------interrupts have priority over all ---------interrupt

►internal, external (Page 279)

►external, internal

331. The ------- can also be used anywhere in the source file to force code at a particular address in the memory.

►.end directive

►.start directive

►.org directive (Page298)

►.label directive

332. : In this figure, the constant value specified by the immediate field is added to the register value, and the resultant is the index of memory location that is referred i.e. Effective Address = A + (content of R) . Identify the addressing mode.

► Displacement (Page 139)

► Immediate

► Indexed

► Relative

333. In which one of the following addressing modes, the operand does not specify an address but it is the actual data to be used.

► Direct

► Indirect

► Immediate click here for detail

► Relative 25

334. : When is the “Divide error interrupt‖ generated?

► When an attempt is made to divide by decimal number

► When an attempt is made to multiply by zero

► When an attempt is made to divide by zero (Page 197)

► When negative number is stored in a register

335. Which one of the following is a term used to describe a storage systems' resilience to disk failure through the use of multiple disks and by the use of data distribution and correction techniques?

► Interrupt handling

► Programmed I/O

► Polling

► RAID click here for detail

336. : _____ is the time for first bit of the message to arrive at the receiver including delays.

► Transmission Time

► Latency

► Transport Latency\

► Time of Flight (Page 388)

337. Falcon-A Simulator loads a FALCON-A binary file with a _____ extension and presents its contents into different areas of the simulator.

► .bin

► .binfa (Page 5)

► .fa

► None of the given

338. In machines where instructions can be executed in parallel or out of order, two additional hazards can occur:

WAW and ----------

- None fo the given
- WAR
- RAW
- RAR

Address	MuxControl	Branch	Br(CON=0)	Br(n=1)	Br(n=0)	End	PCout	LMAR	Control Signals	Branch Address	RTL
300	00	0	0	0	0	0	1	1	…	xxx	MAR← PC: C ← PC + 4;
301	00	0	0	0	0	0	0	0	…	xxx	MBR← M[MAR]: PC ← C;
302	01	1	0	0	0	0	0	0	…	xxx	IR,Micro-PC←MBR<31…27>;
400	00	0	0	0	0	0	0	0	…	xxx	A ← R[rb];
401	00	0	0	0	0	0	0	0	…	xxx	C ← A + R[rc];
402	11	1	0	0	0	1	0	0	…	300	R[ra] ← C; Micro-PC ← 300;

Assume the first control word at address 300. The RTL of this instruction is MAR PC combined with C PC+4. To facilitate these actions the PCout signal bit and the LMAR signal bit are set to one, so that the value of the PC may be written to the internal processor bus and written onto the MAR. The instructions at 300, 301 and 302 form the microcode for instructions fetch. If we examine the RTL we can see all the functionality of the fetch instruction. The value of PC is incremented, the old value of PC is sent to memory, the instruction from the sent address is loaded into memory buffer register. Then the opcode of the fetched instruction is used to invoke the appropriate microroutine.

Alternative approaches to microcoding

- Bit ORing
- Nanocoding
- Writable Microprogram Memory
- Subroutines in Microprogramming

Lecture No. 23

I/O Subsystems

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.1, 8.2

Summary

- Introduction to I/O Subsystems
- Major Components of an I/O Subsystems
- Computer Interface
- Memory Mapped I/O versus Isolated I/O
- Considerations during I/O Subsystem Design
- Serial and Parallel Transfers
- I/O Buses

Introduction to I/O Subsystems

This module is about the computer's input and output. As we have seen in the case of memory subsystems, that when we use the terms " read" and "write", then these terms are from the CPU's point of view. Similarly, when we use the terms "input" and "output" then these are also from the CPU's point of view. It means that when we are talking about an input cycle, then the CPU is receiving data from a peripheral device and the peripheral device is providing data. Similarly, when we talk about an output cycle then the CPU is sending data to a peripheral device and the peripheral device is receiving data. I/O Subsystems are similar to memory subsystems in many aspects. For example, both exchange bits or bytes. This transfer is usually controlled by the CPU. The CPU sends address information to the memory and the I/O subsystems. Then these subsystems decode the address and decide which device should be involved in the transfer. Finally the appropriate data is exchanged between the CPU and the memory or the I/O device.

Memory and I/O subsystems differ in the following ways:

1. Wider range of data transfer speed:

I/O devices can be very slow such as a keyboard in which case the interval between two successive bytes (or keystrokes) can be in seconds. On the other extreme, I/O devices can be very fast such as a disk drive sending data to the CPU or a stream of packets arriving over a network, in which case the interval between two successive bytes can be in microseconds or even nanoseconds. While I/O devices can have such a wide range of data transfer speed compared to the CPU's speed, the case of memory devices is not so. Even if a memory device is slow compared to the CPU, the CPU's speed can be made compatible by inserting wait states in the bus cycle.

2. Asynchronous activity:

Memory subsystems are almost always synchronous. This means that most memory transfers are governed by the CPU's clock. Generally this is not the case with I/O subsystems. Additional signals, called handshaking signals, are needed to take care of asynchronous I/O transfers.

3. Larger degradation in data quality:

Data transferred by I/O subsystems can carry more noise. As an example, telephone line noise can become part of the data transferred by a modem. Errors caused by media defects on hard drives can corrupt the data. This implies that effective error detection and correction techniques must be used with I/O subsystems.

4. Mechanical nature of many I/O devices:

Many I/O devices or a large portion of I/O devices use mechanical parts which inherently have a high failure rate. In case an I/O device fails, interruptions in data transfer will occur, reducing the throughput. As an example, if a printer runs out of paper, then additional bytes cannot be sent to it. The CPU's data should be buffered (or kept in a temporary place) till the paper is supplied to the printer, otherwise the CPU will not be able to do anything else during this time.

To deal with these differences, special software programs called device drivers are made a part of the operating system. In most cases, device drivers are written in assembly language.

You would recall that in case of memory subsystems, each location uses a unique address from the CPU's address space. This is generally not the case with I/O devices. In most cases, a group or block of contiguous addresses is assigned to an I/O device, and data is exchanged byte-by-byte. Internal buffers (memory) within the device store this data if needed.

In the past, people have paid a lot of attention to improve the CPU's performance, as a result of which the performance improvement of I/O subsystems was ignored. (I/O subsystems were even called the "orphans" of computer architecture by some people). Perhaps, many benchmark programs and metrics that were developed to evaluate computer systems focused on the CPU or the memory performance only. Performance of I/O subsystems is as important as that of the CPU or the memory, especially in today's world. For example, the transaction processing systems used in airline reservation systems or the automated teller machines in banks have a very heavy I/O traffic, requiring improved I/O performance. To illustrate this point, look at the following example.

Suppose that a certain program takes 200 seconds of elapsed time to execute. Out of these 200 seconds, 180 seconds is the CPU time and the rest is I/O time. If the CPU performance improves by 40% every year for the next seven years because of developments in technology, but the I/O performance stays the same, let us look at the following table, which shows the situation at the end of each year. Remember that Elapsed time = CPU time + I/O time.

This gives us the I/O time = 200 – 180 = 20 seconds at the beginning, which is 10 % of the elapsed time.

Year #	CPU Time	I/O Time	Elapsed Time	I/O Time x100 % Elapsed Time
0	180	20	200	10 %
1	129	20	149	13.42 %
2	92	20	112	17.85 %
3	66	20	86	23.25 %
4	47	20	67	29.85 %
5	34	20	54	37.03 %
6	24	20	44	45.45 %
7	17	20	37	54.05 %

It can be easily seen that over seven years, the I/O time will become more than 50 % of the total time under these conditions. Therefore, the improvement of I/O performance is as important as the improvement of CPU performance. I/O performance will also be discussed in detail in a later section.

Major components of an I/O subsystem

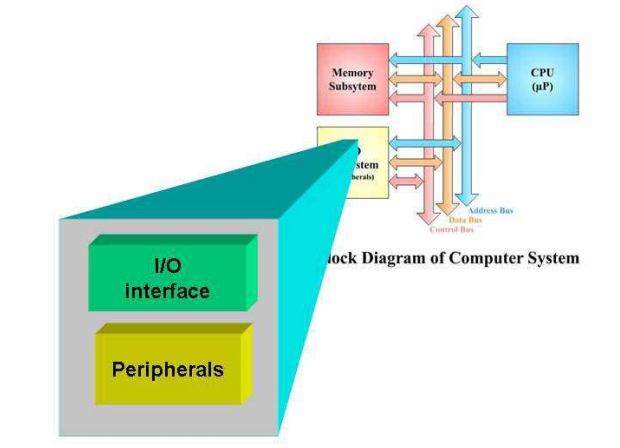
I/O subsystems have two major parts:

- The I/O interface, which is the electronic circuitry that connects the CPU to the I/O device.
- Peripherals, which are the devices used to communicate with the CPU, for example, the keyboard, the monitor, etc.

Computer Interface

A Computer Interface is a piece of hardware whose primary purpose is to connect together any of the following types of computer elements in such a way that the signal levels and the timing requirements of the elements are matched by the interface. Those elements are:

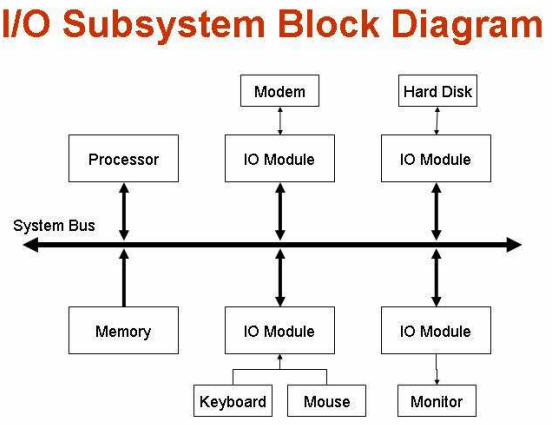
- The processor unit
- The memory subsystem(s)
- Peripheral (or I/O) devices
- The buses (also called "links")



In other words, an interface is an electronic circuit that matches the requirements of the two subsystems between which it is connected. An interface that can be used to connect the microcomputer bus to peripheral devices is called an I/O Port. I/O ports serve the following three purposes:

- Buffering (i.e., holding temporarily) the data to and from the computer bus.
- Holding control information that dictates how a transfer is to be conducted.
- Holding status information so that the processor can monitor the activity of the interface and its associated I/O element.

This control information is usually provided by the CPU and is used to tell the device how to perform the transfer, e.g., if the CPU wants to tell a printer to start a new page, one of the control signals from the CPU can be used for a paper advance command, thereby telling the printer to start printing from the top of the next page. In the same way the CPU may send a control signal to a tape drive connected in the system asking it to activate the rewind mechanism so that the start of the tape is positioned for access by the CPU. Status information from various devices helps the CPU to know what is going on in the system. Once again, using the printer as an example, if the printer runs out of paper, this information should be



sent to the CPU immediately. In the same way, if a hard drive in the system crashes, or if a sector is damaged and cannot be read, this information should also be conveyed to the CPU as soon as possible

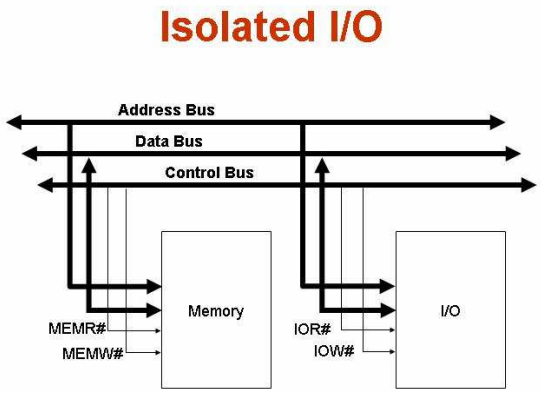
The term "buffer" used in the above discussion also needs to be understood. In most cases, the word buffer refers to I/O registers in an interface where data, status or control information is temporarily stored. A block of memory locations within the main memory or within the peripheral devices is also called a buffer if it is used for temporary storage. Special circuits used in the interfaces for voltage/current matching, at the input and the output, are also called buffers.

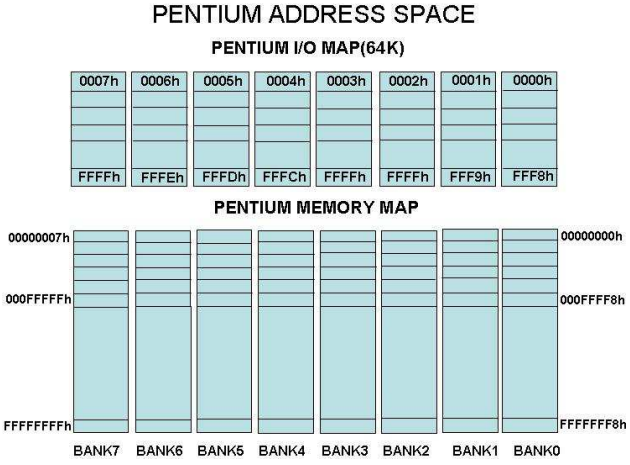
The given figure shows a block diagram of a typical I/O subsystem connected with the other components in a computer. The thick horizontal line is the system bus that serves as a back-bone in the entire computer system. It is used to connect the memory subsystems as well as the I/O subsystems together. The CPU also connects to this bus through a "bus interface unit", which is not shown in this figure. Four I/O modules are shown in the figure. One module is used to connect a keyboard and a mouse to the system bus. A second module connects a monitor to the system bus. Another module is used with a hard disk and a fourth I/O module is used by a modem. All these modules are examples of I/O ports. A somewhat detailed view of these modules is shown in the next figure.

As we already know that the system bus actually consists of three buses, namely the address bus, the data bus and the control bus. These three buses are being applied to the I/O module in this figure. At the bottom, we see a set of data, status and control lines from each "device interface logic" block. Each of these sets connects to a peripheral device. I/O decoding logic is also shown in this figure.

Memory Mapped I/O versus Isolated I/O

Although this concept was explained earlier as well, it will be useful to review it again in this context. In isolated I/O, a separate address space of the CPU is reserved for I/O operations. This address space is totally different from the address space used for memory devices. In other words, a CPU has two distinct address spaces, one for memory and one for input/output. Unique CPU instructions are associated with the I/O space, which means that if those instructions are executing on the CPU, then the accessed address space will be the I/O space and hence the devices mapped on the I/O space.



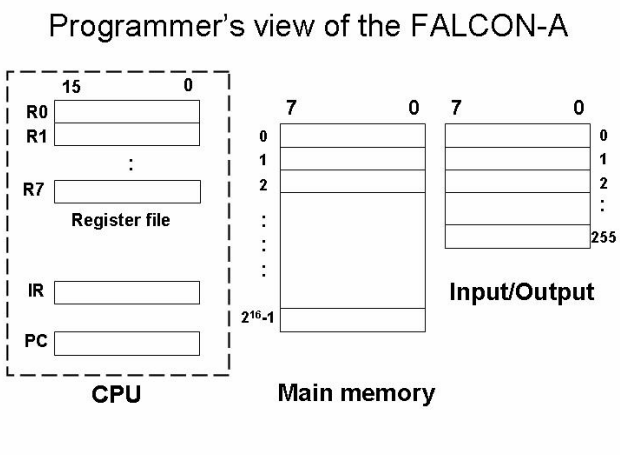


The x86 family with the **in** and the **out** instructions is a well-known example of this situation. Using the **in** instruction, the Pentium processor can receive information from a peripheral device, and using the **out** instruction, the Pentium processor can send information to a peripheral device. Thus, the I/O devices are mapped on the I/O space in case of the Pentium processor. In some processors, like the SRC, there is no separate I/O space. In this case, some address space out of the memory address space must be used to

map I/O devices. The benefit will be that all the instructions which access memory can be used for I/O devices. There is no need for including separate I/O instructions in the ISA of the processor. However, the disadvantage will be that the I/O interface will become complex. If partial decoding is used to reduce the complexity of the I/O interface, then a lot of memory addresses will be consumed. The given figure shows the memory address space as well as the I/O address space for the Pentium processor. The I/O space is of size 64 Kbytes, organized as eight banks of 8 Kbytes each.

A similar diagram for the FALCON-A was shown earlier and is repeated here for easy reference.

The next question to be answered is how the CPU will differentiate between these two address spaces. How will the system components know whether a particular transfer is meant for memory or an I/O device? The answer is simple: by using signals from the control bus, the CPU will indicate which address space is meant during a particular transfer. Once again, using the Pentium as an example, if the **in** instruction is executing on the processor, the IOR# signal will become active and the MEMR# signal will be deactivated. For a **mov** instruction, the control logic will activate the MEMR# signal instead of the IOR# signal.



Considerations during I/O Subsystem Design

Certain things must be taken care of during the design of an I/O subsystem.

Data location:

The designer must identify the device where the data to be accessed is available, the address of this device and how to collect the data from this device. For example, if a database needs to be searched for a record that is stored in the fourth sector of the second track of the third platter on a certain hard drive in the system, then this information is related to data location. The particular hard drive must be selected out of the possibly many hard drives in the system, and the address of this record in terms of platter number, track number and sector number must be given to this hard drive.

Data transfer:

This includes the direction of transfer of data; whether it is out of the CPU or into the CPU, whether the data is being sent to the monitor or the hard drive, or whether it is being received from the keyboard or the mouse. It also includes the amount of data to be transferred and the rate at which it should be transferred. If a single mouse click is to be transferred to the CPU, then the amount of data is just one bit; on the other hand, a block of data for the hard drive may be several kilo bytes. Similarly, the rate of the transfer of data to a printer is very different from the transfer rate needed for a hard drive.

Data synchronization:

This means that the CPU should input data from an input device only when the device is ready to provide data and send data to an output device only when it is ready to receive data.

There are three basic schemes which can be used for synchronization of an I/O data transmission:

- Synchronous transmission
- Semi-synchronous transmission
- Asynchronous transmission

Synchronous transmission:

This can be understood by looking at the waveforms shown in Figure A. M stands for the bus master and S stands for the slave device on the bus. The master and the slave are assumed to be permanently connected together, so that there is no need for the selection of the particular slave device out of the many devices that may be present in the system. It is also assumed that the slave device can perform the transfer at the speed of the master, so no handshaking signals are needed.

At the start of the transfer operation, the master activates the Read signal, which indicates to the slave that it should respond with data. The data is provided by the slave, and the master uses the Enable signal to latch it. All activity takes place synchronously with the system clock (not shown in the figure). A familiar example of synchronous transfer is a register-to-register transfer within a CPU.

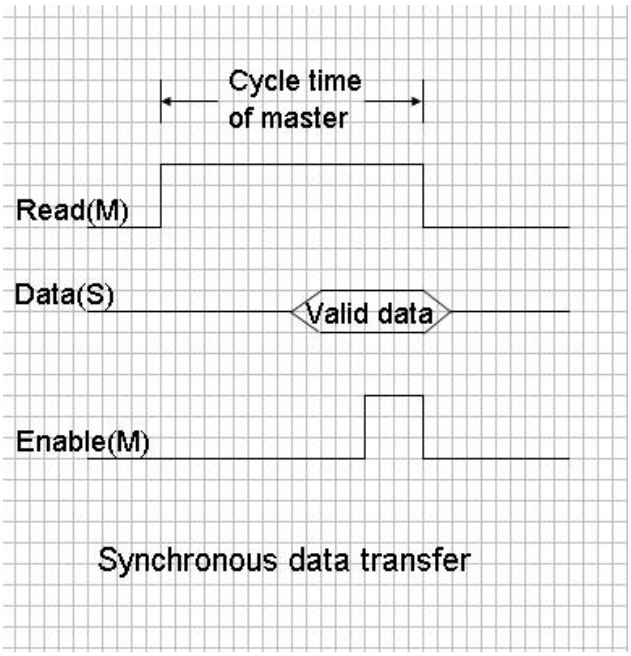


Figure A

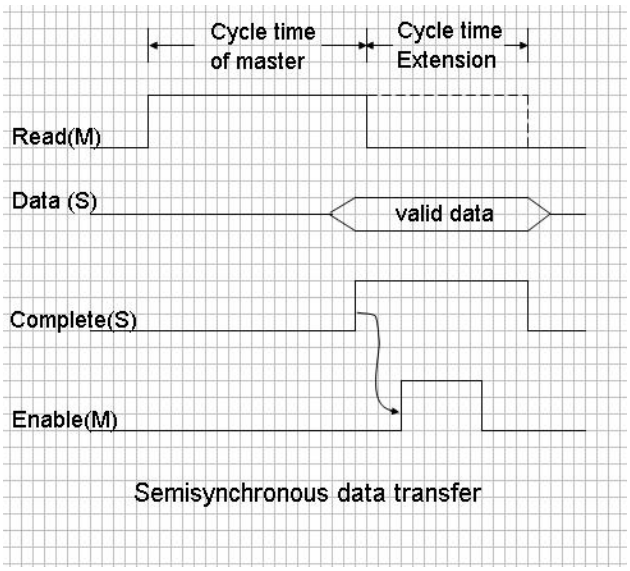


Figure B

Semi-synchronous transmission:

Figure B explains this type of transfer. All activity is still synchronous with the system clock, but in some situations, the slave device may not be able to provide the data to the master within the allotted time. The additional time needed by the slave, can be provided by adding an integral number of clock periods to **Figure A** the master's cycle time.

The slave indicates its readiness by activating the complete signal. Upon receiving this signal, the master activates the Enable signal to latch the data provided by the slave. Transfers between the CPU and the main memory are examples of semi-synchronous transfer.

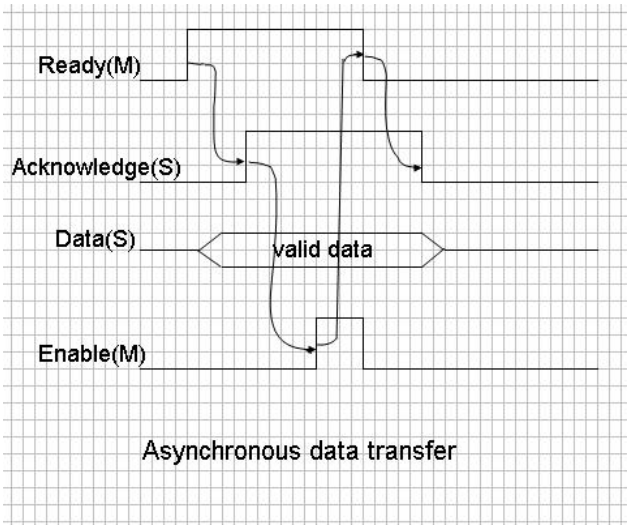


Figure C

Asynchronous transmission:

This type of transfer does not require a common clock. The master and the slave operate at different speeds. Handshaking signals are necessary in this case, and are used to coordinate the data transfer between the master and the slave as shown in the Figure C. When the master wants

to initiate a data transfer, it activates its Ready signal. The slave detects this signal, and if it can provide data to the master, it does so and also activates its Acknowledge signal. Upon receiving the Acknowledge signal, the master uses the Enable signal to latch the incoming data .The master then deactivates its Ready line, and in response to it, the slave removes its data and deactivates its Acknowledge line.

In all the three cases discussed above, the waveforms correspond to an “input” or a “read” operation. A similar explanation will apply to an “output” or a “write” operation. It should also be noted that the latching of the incoming data can be done by the master either by using the rising edge of the Enable signal or by using its falling-edge. This will depend on the way the intermediate circuitry between the master and the slave is designed.

Serial and Parallel Transfers

There are two ways in which data can be transferred between the CPU and an I/O device: serial and parallel.

Serial Transfer, or serial communication of data between the CPU and the I/O devices, refers to the situation when all the data bits in a "piece of information", (which is a byte or word mostly), are transferred one bit at a time, over a single pair of wires. Advantages:

- Easy to implement, especially by using UARTs[7] or USARTs[8].
- Low cost because of less wires.
- Longer distance between transmitter and receiver. Disadvantages:
- Slow by its very nature.
- Inefficient because of the associated overhead, as we will see when we discuss the serial wave forms.

Parallel Transfer, or parallel communication of data between the CPU and the I/O devices, refers to the situation when all the bits of data (8 or 16 usually), are transferred over separate lines simultaneously, or in parallel. Advantages:

- Fast (compared to serial communication) Disadvantages:
- High cost (because of more lines).
- Cost increases with distance.
- Possibility of interference (noise) increases with distance.

Remember that the terms "serial" and "parallel" are with respect to the computer I/O ports and not with respect to the CPU. The CPU always transfers data in parallel.

Types of serial communication

There are two types of serial communication:

Asynchronous:

- Special bit patterns separate the characters.
- "Dead time" between characters can be of any length.
- Clocks at both ends need not have the same frequency (within permissible limits).

7 Universal Asynchronous Receiver Transmitter.
8 Universal Synchronous Asynchronous Receiver Transmitter.

Synchronous:

- Characters are sent back to back.
- Must include special "sync" characters at the beginning of each message.
- Must have special "idle" characters in the data stream to fill up the time when no information is being sent.
- Characters must be precisely spaced.
- Activity at both ends must be coordinated by a single clock. (This implies that the clock must be transmitted with data).

The "maximum information rate" of a synchronous line is higher than that of an asynchronous line with the same "bit rate", because the asynchronous transmission must use extra bits with each character. Different protocols are used for serial and parallel transfer. A protocol is a set of rules understood by both the sender and the receiver. In some cases, these protocols can be predefined for a certain system. As an alternate, some available standard protocols can be used.

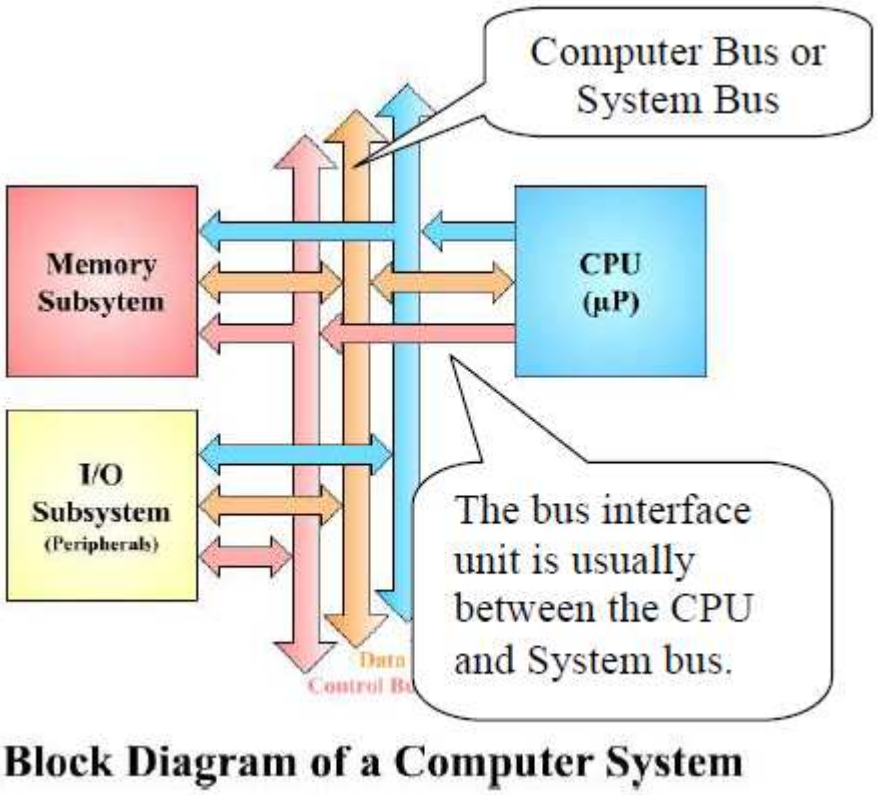


Figure 1

Error conditions related to serial communication
(Some related to synchronous transmission, some to asynchronous, and some to both).

- Framing Error: is said to occur when a 0 is received instead of a stop bit (which is always a 1). It means that after the detection of the beginning of a character with a start bit, the appropriate number of stop bits was not detected. [A]
- Parity Error: is said to occur when the parity* of the received data is not the same as it should be. [B] (PARITY is equivalent to the number of 1's; it is either EVEN or ODD. A PARITY BIT is an extra bit added to the data, for the purpose of error detection and correction. If even parity is used, the parity bit is set so that the total number of 1's, including the parity bit, is even. The same applies to odd parity.)

- Overrun Error: means that the prior character that was received, was not yet read from the USART's "receive data register" by the CPU, and is overwritten by the new received character. Thus the first character was lost, and should be retransmitted. [A]

- Under-run Error: If a character is not available at the beginning of an interval, an under-run is said to occur. The transmitter will insert an idle character till the end of the interval. [S]

I/O Buses

The block diagram of a general purpose computer system that has been referred to repeatedly in this course has three buses in addition to the three most important blocks. These three buses are collectively referred to as the system bus or the computer bus[9]. The block diagram is repeated here for an easy reference in Figure 1.

Another organization that is used in modern computers is shown in Figure 2. It has a memory bus for connecting the CPU to the memory subsystem. This bus is separate from the I/O bus that is used to connect peripherals and I/O devices to the system.

Examples of I/O buses include the PCI bus and the ISA bus. These I/O buses provide an "abstract interface" that can be used for interfacing a large variety of peripherals to the system with minimum hardware. It is also possible to standardize I/O buses, as done by several agencies, so that third party manufacturers can build add-on sub systems for existing architectures.

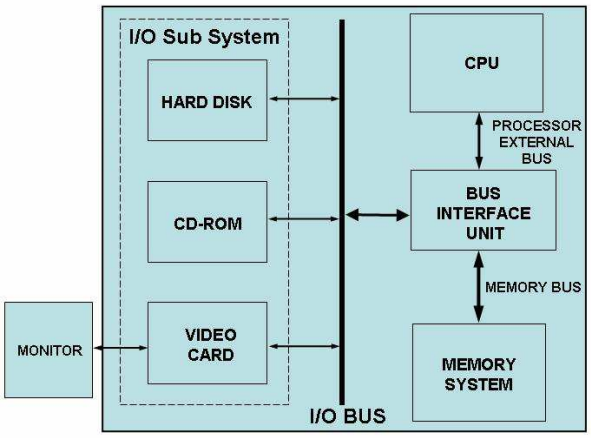


Figure 2

The location of these I/O buses may be different in different computers.

Earlier generation computers used a single bus over which the CPU could communicate with the memory as well as the I/O devices. This meant that the bandwidth of the bus was shared between the memory and I/O devices.

However, with the passage of time, FIGURE 1 computer architects drifted towards separate memory and I/O buses, thereby giving more flexibility to users wanting to upgrade their existing systems. A main disadvantage of I/O buses (and the buses in general) is that every bus has a fixed bandwidth which is shared by all devices on the bus. Additionally, electrical constraints like transmission line effects and bus length further reduce the bandwidth. As a result of this, the designer has to make a decision whether to sacrifice interface simplicity (by connecting more devices to the bus) at the cost of bandwidth, or connect fewer devices to the bus and keep things simple to get a better bandwidth. This can be explained with the help of an example.

Example # 1

Problem statement:

Consider an I/O bus that can transfer 4 bytes of data in one bus cycle. Suppose that a designer is considering to attach the following two components to this bus:

[9] In some cases, the external CPU bus is the same as the dedicated systems. However, for most systems, there is a "bus interface unit" between the CPU and the system bus. The bus interface unit is not shown in the figure.

Hard drive, with a transfer rate of 40 Mbytes/sec Video card, with a transfer rate of 128 Mbytes/sec. What will be the implications?

Solution:

The maximum frequency of the bus is 30 MHz[10]. This means that the maximum bandwidth of this bus is 30 x 4 = 120 Mbytes/sec. Now, the demand for bandwidth from these two components will be 128 + 40 =168 Mbytes/sec which is more than the 120 Mbytes/sec that the bus can provide. Thus, if the designer uses these two components with this bus, one or both of these components will be operating at reduced bandwidth.

Bus arbitration:

Arbitration is another issue in the use of I/O buses. Most commercially available I/O buses have protocols defining a number of things, for example how many devices can access the bus, what will happen if multiple devices want to access the bus at the same time, etc. In such situations, an "arbitration scheme" must be established. As an example, in the SCSI[11] specifications, every device in the system is assigned an ID which identifies the device to the "bus arbiter". If multiple devices send a request for the bus, the device with the highest priority will be given access to the bus first. Such a scheme is easy to implement because the arbiter can easily decide which device should be given access to the bus, but its disadvantage is that the device with a low priority will not be able to get access to the bus[12]. An alternate scheme would be to give the highest priority to the device that has been waiting for the longest time for the bus. As a result of this arbitration, the access time, or the latency, of such buses will be further reduced. Details about the PCI and some other buses will be presented in a separate section.

Example # 2

Problem statement:

If a bus requires 10 nsec for bus requests, 10 nsec for arbitration and the average time to complete an operation is 15 nsec after the access to the bus has been granted, is it possible for such a bus to perform 50 million IOPS?

Solution:

For 50 million IOPS, the average time for each IOP is 1 / (50 x 10^6) =20 nsec. Given the information about the bus, the sum of the three times is 10 + 10 + 15 = 35 nsec for a complete I/O operation. This means that the bus can perform a maximum of 1 / (35 x 10^{-9}) = 28.6 million IOPS.
Thus, it will not be able to perform 50 million IOPS.

---

10 These numbers correspond to an I/O bus that is relatively old. Modern systems use much faster buses than this.

11 Small Computer System Interface.

12 Such a situation is called "starvation".

Lecture No. 24
Designing Parallel Input and Output Ports

Reading Material

Handouts Slides

Summary

- Designing Parallel I/O Ports
- Practical Implementation of the SAD
- NUXI Problem
- Variation in the Implementation of the Address Decoder
- Estimating the Delay Interval

Designing Parallel I/O Ports

This section is about designing parallel input and output ports. As you already know from the previous discussion, an interface that is used to connect the computer bus with I/O devices is called an I/O port. This I/O port can be connected directly to the computer bus (also called the system bus) or through an intermediate bus called the I/O bus. This intermediate bus is also called the expansion bus or the peripheral bus. In any case, the following general information about I/O bus cycles on a typical CPU should be kept in mind: At the start of a particular bus cycle (which will be an I/O bus cycle in this case), the CPU places an address on its address bus. This address will identify the I/O device to be involved in the transfer. After some time the CPU will activate certain control signals, which will indicate whether the particular I/O bus cycle, is an I/O read or an I/O write cycle. Based on these control signals, in case of I/O read cycle, the CPU will be expecting data from the selected input device over the data bus, and for an I/O write cycle the CPU will provide data to the selected device over the data bus. At the end of this I/O bus cycle, the address (and data) information will be removed from the buses and the control signals will be reset. It can be easily understood from this discussion that we must match the timing requirements of the I/O ports to be designed with the timing parameters of the given CPU. Additionally, the voltage and current requirements of the I/O ports must be matched with the voltage and current specifications of the CPU. For simplicity, we ignore the voltage and current matching details in this discussion and only focus on the logic levels and timing aspects of the design. Voltage and current related discussions are the topic of an electronics course.
Thus, there are two important functions which should be built into I/O ports.

1. Address decoding
2. Data isolation for input ports or data capturing for output ports.

1. Address decoding: Since every I/O port has a unique identifier associated with it, (which is called its ***address***, and no other port in the system should have the same address), by monitoring the system address bus, the I/O port knows when it is its turn to participate in a transfer. At this time, the address decoder within the I/O port generates an asserted output which can be applied to the enable input of tri-state buffers in input ports or the latch enable input of latches in output ports.

Our definition of an address decoder:

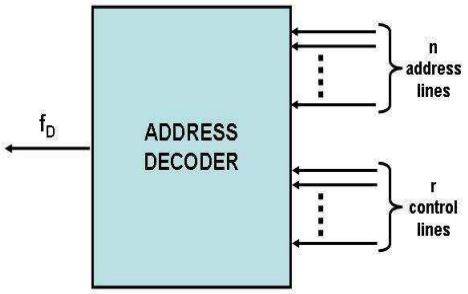
An "Address Decoder" is a combinational (logic) circuit with ***n*** + ***r*** inputs and a single output, where

n = the number of address lines into the decoder, and

r = the number of control lines into the decoder.

The output f_D is active only when the corresponding address is present on the ***n*** address lines and the corresponding ***r*** control lines hold the "proper" (active or inactive) value. f_D is inactive for all other situations.

Block diagram of an address decoder



Suggestions for address decoder design:

1.1 Start by thinking of the address decoder as a "big AND gate". We will call this a "skeleton address decoder" or SAD. The output of the SAD will be active only when the correct address is present on the system address bus and the relevant control bus signals hold the proper values. At all other times, the output of the SAD should be deactivated.

1.2 Always write the port address of the port to be designed in binary. Associate the CPU's address lines with each bit. Those lines which are zero will be inverted before being fed into the "big AND gate"; other address lines will not be inverted.

1.3 List the relevant control signals for the system to which the port is to be attached. If the "proper" value of the signal is 0, it should be inverted before applying to the SAD, otherwise it is fed directly into the SAD.

1.4 Determine whether the decoder output should be active high or low. This will depend on the type of latch or buffer used in the design. If an active low decoder output is needed, invert the output from the "big AND gate".

1.5 Once the logic for the address decoder is established, the SAD can be implemented using any of the available methods of logic design. For example, HDL code in Verilog or VHDL can be generated and the address decoder can be implemented using PLDs. Alternately, the SAD can be implemented using SSI building blocks.

2. Data isolation or capturing: For input ports, the incoming data should be placed on the data bus only during the I/O read bus cycle. At all other times, this data should be isolated from the data bus otherwise it will cause "bus contention". Tri-state buffers are used for this purpose. Their input lines are connected to the peripheral device supplying data and their output lines are connected to the data bus. The common enable line of such buffers is driven with the output of the SAD. If this enable is active low, the output of the big AND gate in the SAD should be inverted, as described earlier.

For output ports, data is made available for the peripheral device at the data bus during the I/O write bus cycle. During other bus cycles, this data will be removed from the data bus by the processor. Latches (or registers) are used for this purpose. Their input lines are connected to the system data bus and their output lines are connected to the peripheral device receiving data. The common clock (or latch enable) line of such latches is driven with the output of the SAD. If this clock is active low, the output of the big AND gate in the SAD should be inverted.

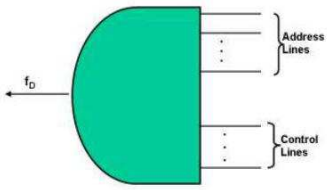
Example # 1

Problem Statement:

Design a 16-bit parallel output port mapped on address DEh of the I/O space of the FALCON-A CPU.

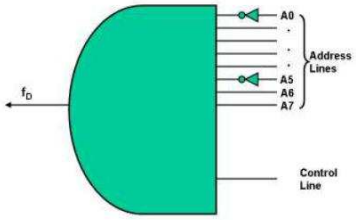
Solution:

Using the guidelines mentioned above, we start with a "big AND gate" (SAD) and write the address to be decoded (DEh) in binary.



Thus, DEh → 1101 1110 b. Associating one CPU address line with each bit, we get A0 = 0, A1=1, etc as shown in the table below.

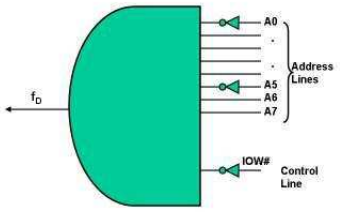
Because the I/O space on the FALCON-A is only 256 bytes, address lines A15 .. A8 are don't cares, and will not be used in this design.



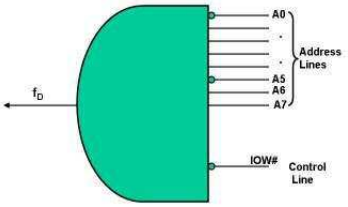
1	1	0	1	1	1	1	0
A7	A6	A5	A4	A3	A2	A1	A0

Thus, A0 and A5 will be applied to the "big AND gate" after inversion. The remaining address lines will be connected directly to the inputs of the SAD.

Next, we look at the relevant control signals. The only signal which should be used in this case is IOW#. A logic 0 (zero) on this line indicates that it is active. Thus, it should be inverted before being applied to the input of the SAD.



We can easily see that our SAD intuitively conforms to the way we defined an address decoder. Its output is a 1 only when the address (xxxx xxxx 1101 1110 b) is present on the FALCON-A's address bus during an I/O write cycle (By the way, this will take place when the instruction **out reg, addr** with **addr=DEh or 222d** is executing on the FALCON-A). At all other times, its output will be inactive.



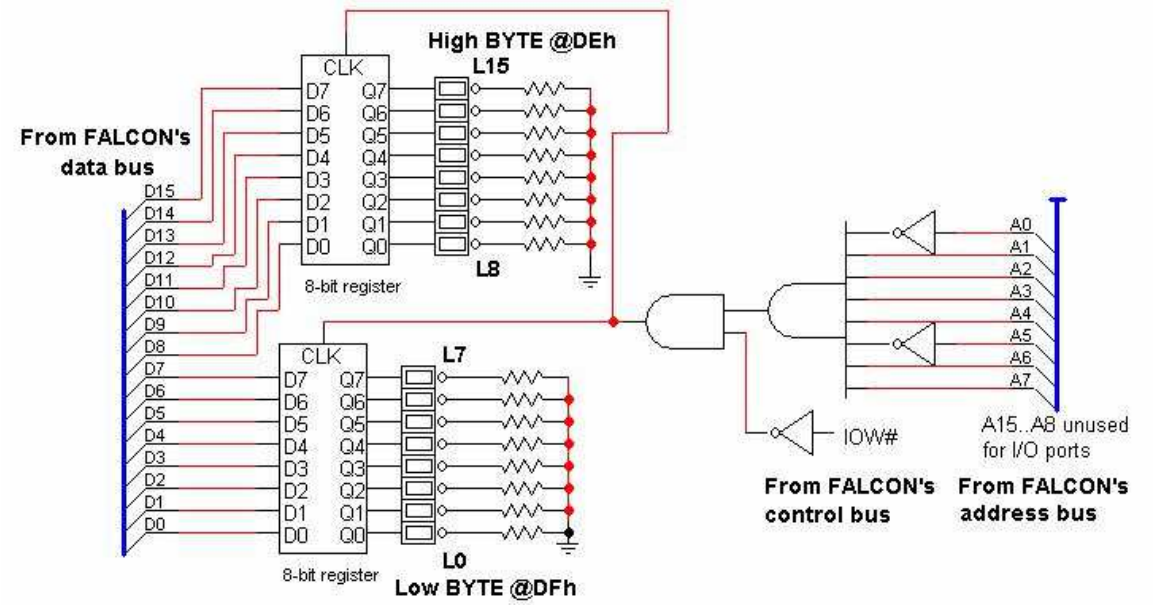
To make things simple, we use a circle (or a bubble) to indicate an inverter, as shown .Since this is a 16-bit output port, we will use two 8-bit registers to capture data from the FALCON-A's data bus. The output of the SAD will be connected to the enable inputs of the two registers. The D-inputs of the registers will be connected to the data bus and the Q outputs of the registers will be connected to the peripheral device.

Practical implementation of the SAD

Our SAD in this design is an AND gate with 9 inputs. Using SSI chips, we can implement this SAD using an 8-input AND gate and a 2-input AND gate as shown in the figure shown below.

Displaying output data using LED branches:

An "LED branch" is a combination of a resistor and a light emitting diode (LED) in series. Sixteen LED branches can be used to display the output data captured by the registers as shown in the figure below.



A 16-bit parallel output port for the FALCON-A at address DEh and DFh

Example # 2

Problem statement:

Given a 16-bit parallel output port attached with the FALCON-A CPU as shown in the figure. The port is mapped onto address DEh of the FALCON-A's I/O space. Sixteen LED branches are used to display the data being received from the FALCON-A's data bus. Every LED branch is wired in such a way that when a 1 appears on the particular data bus bit, it turns the LED on; a 0 turns it off. Which LEDs will be ON when the instruction

out r2, 222 [13]

executes on the CPU? Assume r2 contains 1234h.

Solution:

Since r2 contains 1234h, the bit pattern corresponding to this value will be sent out to the output port at address 222 (or DEh). This is the address of the output port in this example. Writing the bit pattern in binary will help us determine the LEDs which will be ON.

Now 1234h gives us the following bit associations with the data bus

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

MSB at address DEh LSB at address DFh

Note that the 8-bit register which uses lines D15 .. D8 of the FALCON-A's data bus is actually mapped onto address DEh of the I/O space.

13 Depending on the way the assembler is written, the syntax of the **out** instruction may allow only the decimal form of the port address, or only the hexadecimal form, or both. Our version of the assembler for the FALCON-A allows the decimal form only. It also requires that the port address be aligned on 16-bit "word boundaries", which means that every port address should be divisible by 2.

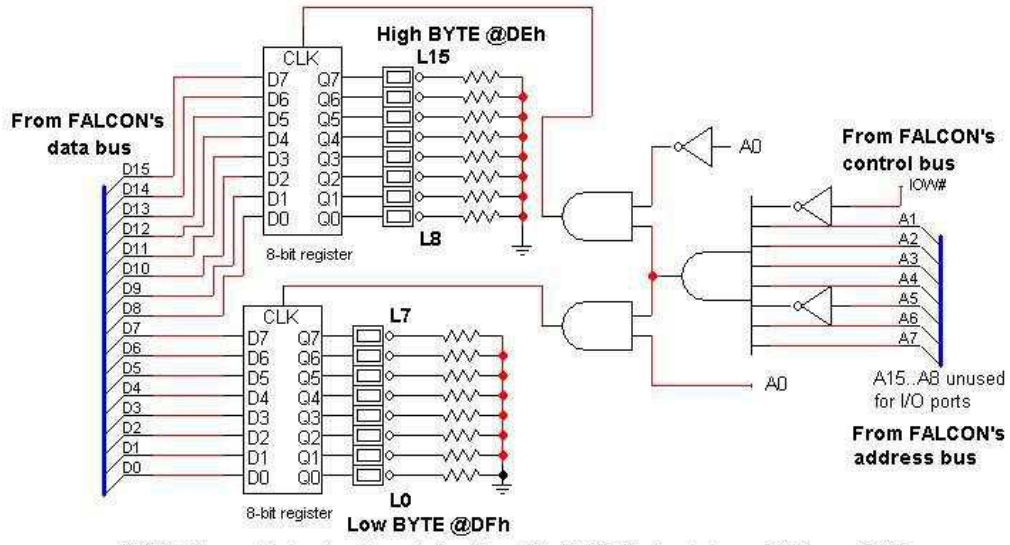
This is because the architect of the FALCON-A had chosen a "byte-wide" (i.e., x8) organization of the address space, a 16-bit data bus width, and the "big-endian" data format at the ISA design stage. Additionally, data bus lines D15...D8 will transfer the data byte of higher significance (MSB) using address DEh, and D7...D0 will transfer the data byte of lower significance (LSB) using address DFh. Thus the LEDs at L12, L9, L5, L4 and L2 will turn on.

The NUXI Problem

It can be easily understood from the previous example that the big-endian format results in the least significant byte being transferred over the most significant side of the data bus, and vice versa. The situation will be exactly opposite when the little-endian format is used. In this case, the least significant byte will be transferred over the least side of the data bus. Now imagine a computer using the little-endian format exchanging data with a computer using the big-endian format over a 16-bit parallel port. (this may be the case when we have a network of different types of computer, for example). The data transmitted by one will be received in a "swapped" form by the other, eg., the string "UN" will be received as "NU" and the string "IX" will be received as "XI". So UNIX changes to NUXI --- hence the name NUXI problem. Special software is used to resolve this problem.

Variation in the Implementation of the Address Decoder

The implementation of the address decoder shown in Example #1(lec24) assumes that the FALCON-A does not allow the use of some part of its data bus during an I/O (or memory) transfer. Another restriction that was imposed by the assembler was that all port addresses should be divisible by 2. This implies that address line A0 will always be zero. If the FALCON-A architect had allowed the use some of part of its data bus (eg, 8-bits) during a transfer, the situation would be different.



A 16-bit parallel output port for the FALCON-A at address DEh and DFh

The logic diagram shown in the next figure is a 16-bit parallel output port at the same address (DEh) for the FALCON-A assuming that part of its data bus (D15..D8) or (D7..D0) can be used independently during an I/O transfer. Note that the enable inputs of the two 8-bit registers are not connected together in this case. Moreover, since the 16-bit port uses two addresses, address line A0 will be at a logic 0 for address DEh, and at a logic 1 for address DFh. This means that it cannot be used at the input of the big AND gate. So, A0 has been used in a different position with the two 2-input AND gates. The 2-input AND gate where A0 is applied after inversion will generate a 1 at its output when A0 = 0. Thus, this output will enable the 8-bit register mapped on

the even address DEh. In case of the other AND gate, A0 is not inverted. So the corresponding 8-bit register will be mapped on the odd address DFh. The input that became available after removing A0 from its old position can be used for the IOW# control signal. The rest of the circuit is the same as it was in the previous figure.

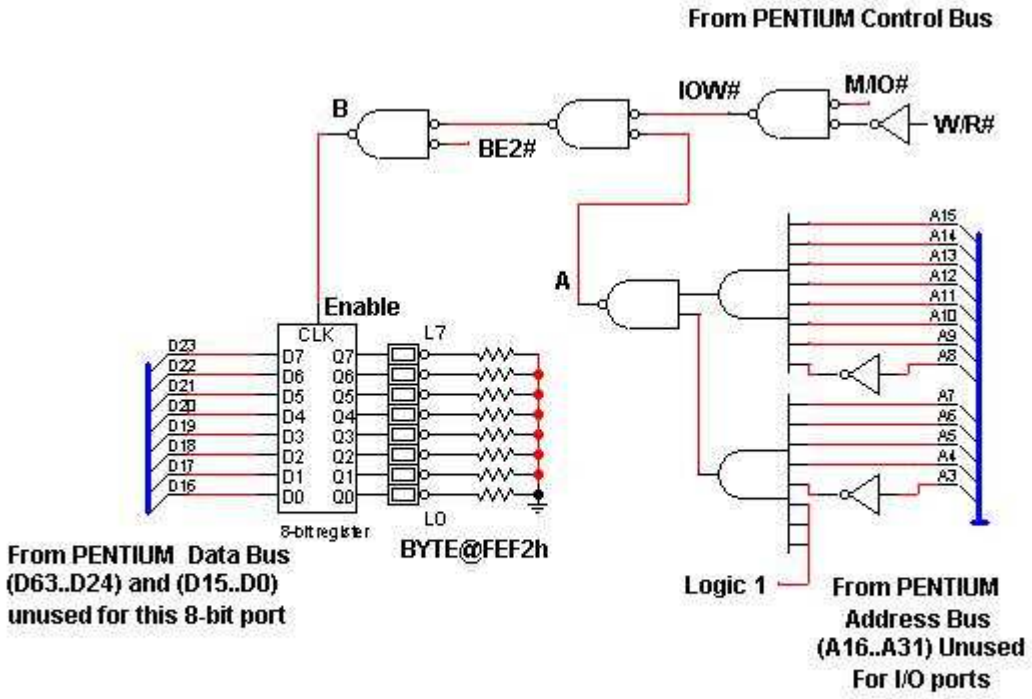
We can understand from the above discussion that the decisions made at the time of ISA design have a strong bearing on the implementation details and the working of the computer. Suppose we assume that the assembler developer had decided not to restrict the port addresses to even values, then what will be the implications?

As an example, consider the execution of the instruction **out r2, 223** assuming r2 contains 1234h. This is a 16-bit transfer at address 223 (DFh) and 224 (E0h).

For the output port (shown in the first figure) where the CPU does not allow the use of some part of its data bus in a transfer, none of the registers will be enabled as a result of this instruction because the output of the 8-input AND gate will be a zero for both addresses DFh and E0h. Thus, that output port cannot be used.

In the second figure, where the CPU has allowed to use a portion of its data bus in an I/O transfer, the register at the address DEh will not be enabled. The CPU will send the high data byte(12h) to the register at the address DFh (because it will be enabled at that time due to the address DFh) over data lines D7…D0. The fact that data lines D7…D0 should be used for the transfer of high byte, will be taken care of by the hardware, internal to the CPU.

Now the question is where the low data byte (i.e. 34h) present at D15…D8 data lines would be placed? If there exists an output port at address E0h in the system, then 34h will be placed there **(in the next bus cycle),** otherwise it will be lost. Again, it is the CPU's responsibility to check whether the next address in the system exists or not and if exists then enable that port so that the low byte of data can be placed there.



An 8-bit Parallel Output Port for the PENTIUM Processor at address FEF2h of the I/O space

A possible option for the architect in this case would be to revisit the design steps and allow the use of part of the CPU registers (or at least for some of them) for I/O transfers. The logic diagram shown below shows an 8-bit parallel output port at address FEF2h of the Pentium's I/O address space. Since the Pentium allows the use of some part of its data bus during a transfer, we

can use the BE2# signal in the address decoder to enable the 8-bit register. The following instructions will access this output port.

```
mov dx, 0FEF2h
mov al, 12h
out dx, al
```

The Pentium **does** allow the use of some part of its 32-bit accumulator register EAX. In case only 8-bits are to be transferred, register AL can be used, as shown in the program fragment above. The data byte 12h will be sent to the 8-bit register over lines D23..D16. Since 12h corresponds to 0001 0010 in binary, this will cause the LEDs L4 and L1 to turn on.

Example # 3

Problem statement:

Write an assembly language program to turn on the 16 LEDs one by one on the output port of Example #1(lec24). Each LED should stay on for a noticeable duration of time. Repeat from the first LED after the last LED is turned on.

Solution:

The solution is shown in the text box with a filename: Example_3.asmfa. The working of this program is explained below:

The first two instructions turn all the LEDs off by sending a 0 to each bit of the output port at address 222.

```
        mov r1,0
        out r1,222
```

Then a 1 is sent to L0 causing it to turn on, and the program enters a loop which executes 15 times to cause the other LEDs (L1 through L15) to turn on, one by one in sequence. Register r5 is being used as loop counter. The following three instructions introduce a delay between successive bit patterns sent to the output port, so that each LED stays on for a noticeable duration of time.

```
delay1: movi r2,0
again1: subi r2,r2,1
        jnz r2,[again1]
```

```
; filename: Example_3.asmfa
;
;ALL LEDS ARE turned Off initially
;
        movi r1,0
        out r1,222
;
;First LED will be turned on each time
;
start:  movi r1,1
        out r1,222
;
        movi r5,15
;
;DELAY LOOP
;
delay1: movi r2,0
again1: subi r2,r2,1
      jnz r2, [again1]
;
        movi r3,0       ; TURN OFF ALL LEDS
        out r3,222
;
delay2: movi r2,0
again2: subi r2,r2,1
         jnz r2, [again2]
;
        shiftl r1,r1,1    ; next LED ON
        out r1,222
        subi r5,r5,1
        jnz r5, [delay1]
        jump [start]
        halt
```

Starting with a value of 0 in r2[14], this value is decremented to FFFFh when the again1 loop is entered. The jnz instruction will cause r2 to decrement again and again; thereby executing the loop 65,535 times. An estimate of the delay interval is presented at the end of this section.

After this delay, all the LEDs are turned off, and a second delay loop executes. Finally, the next LED on the left, in sequence, is turned on by the following two instructions:

[14] This is necessary because the immediate operand with the **movi** instruction of the FALCON-A has a range of 0h to FFh. This will not give us the large loop counter that we need here. So we use the above software trick. An alternate way would be to use nested loops, but that will tie up additional CPU registers.

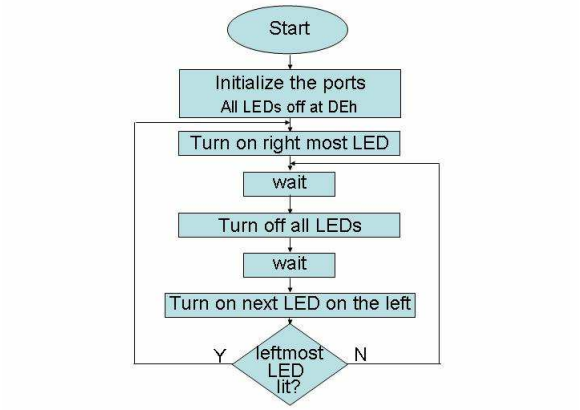
```
shiftl r1,r1,1
out r1, 222
```

After the left most LED is turned on, the process starts all over again because of the last jump instruction. The outermost loop executes indefinitely.

Estimating the Delay Interval

To make things simple, assume that the FALCON-A is operating at a clock frequency of 1 MHz. Also, assume that the **subi** and the **jnz** instructions take 3 and 4 clock periods, respectively, to execute. Since these two instructions execute 65,535 times each, we can use the following formula to compute the execution time of this loop:

ET = CPI x IC x T = CPI x IC / f
where
CPI = clocks per instruction
IC = instruction count
T = time period of the clock,
And
f = frequency of the clock.



Using the assumed values, we get

ET = (3+4) x 65535 / (1x10^6) = 0.459 sec

Since the **movi r2, 0** instruction executes only once, the time it takes to execute is negligible and has been ignored in this calculation.

Lecture No. 25

Input Output Interface

Reading Material

Handouts Slides

Summary

- Designing a Parallel Input Port
- Memory Mapped I/O Ports
- Partial Decoding and the "wrap around" Effect
- Data Bus Multiplexing
- A generic I/O Interface
- The Centronics Parallel Printer Interface

Designing a parallel input port

The following example illustrates a number of important concepts.

Example # 1

Problem statement:

Design an 16-bit parallel input port mapped on address 7Eh of the I/O space of the FALCON-A CPU.

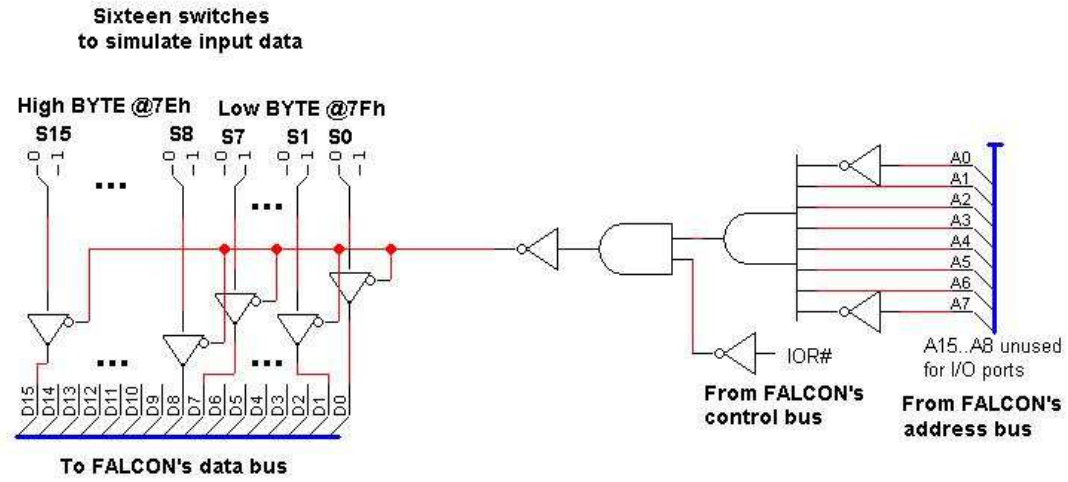
Solution:

The process of designing a parallel input port is very similar to the design of a parallel output port except for the following differences:

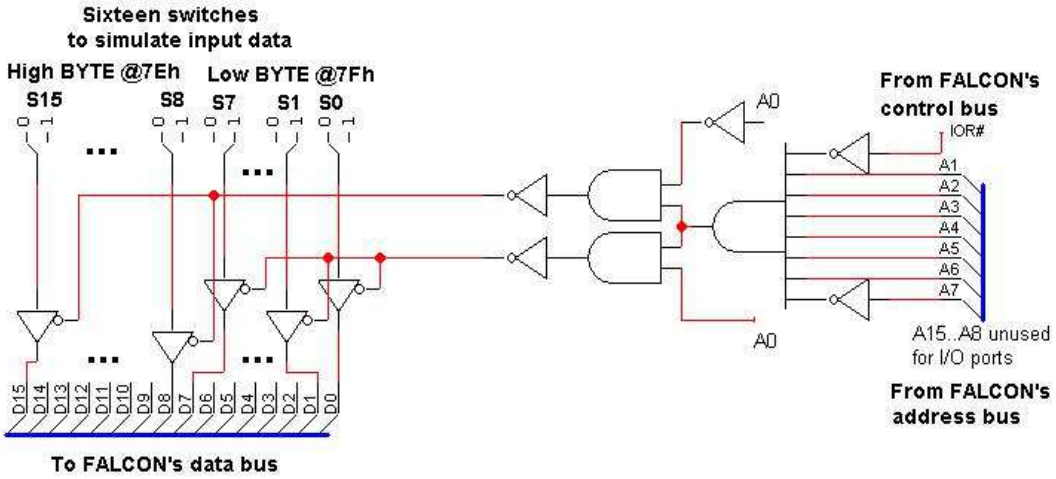
1. The address in this case is 7Eh, which is different from the previous value. Hence, the address decoder will have the inputs A7 and A0 inverted, while the other address lines at its input will not be inverted.
2. Control bus signal IOR# will be used instead of the signal IOW#.
3. A set of sixteen tri-state buffers will be used for data isolation. Their common enable line will be connected to the output of the big AND gate (in the figure, f_D is being inverted because Enable is active low). The input of these buffers can be connected to the input device and the output is connected to the FALCON-A's data bus.

In this example, switches S15...S0 are used to simulate the input data. The complete logic circuit is shown in the next two figures.

In the second figure, the CPU is assumed to allow the use of some part of its data bus during a transfer, while in the first figure it is not allowed.



A 16-bit parallel input port for the FALCON-A at address 7Eh and 7Fh



A 16-bit parallel input port for the FALCON-A at address 7Eh and 7Fh

Example # 2

Problem statement:

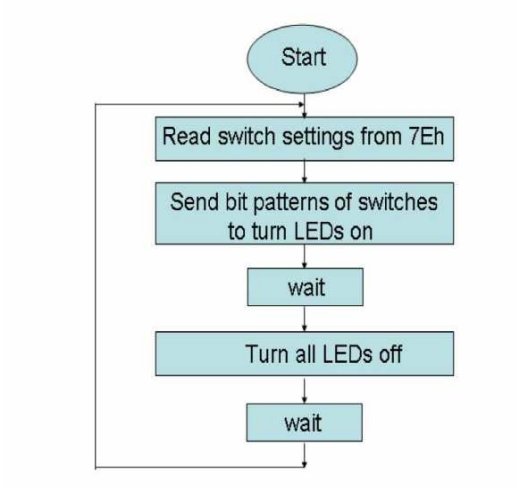
Given a FALCON-A processor with a 16-bit parallel input port at address 7Eh and a 16-bit parallel output port at address DEh. Sixteen LED branches are used to display the data at the output port and sixteen switches are used to send data through the input port. Write an assembly language program to continuously monitor the input port and blink the LED or LED(s) corresponding to the switch (es) set to logic 1. For example, if S0 and S2 are set to 1, then only the LEDs L0 and L2 should blink. If S7 is also set to logic 1 later, then L7 should also start blinking.

Solution:

The program is shown in the text box with filename: Example_2. It works as explained below;

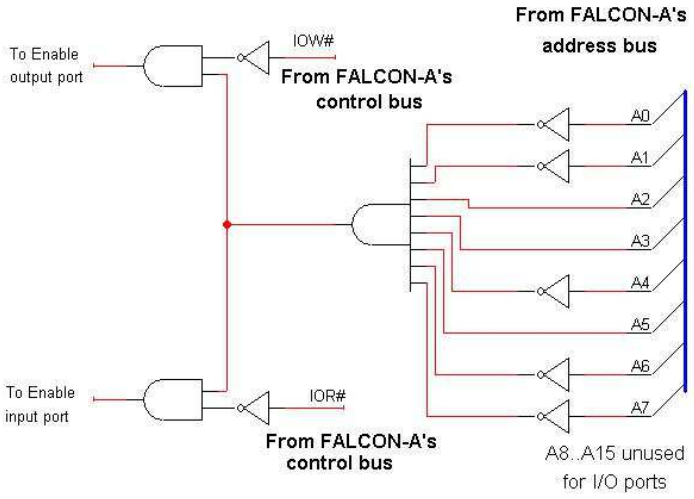
The first two instructions read the input port at address 7Eh and send this bit pattern to the output port at address DEh. This will cause the LEDs corresponding to the switches that are set to a 1 to turn on. Next, the program waits for a suitable amount of time, and then turns all LEDs off and waits again.

After the second wait, the program reads the input port again. The LEDs that will be turn on at the output port will now be according to the new switch settings at the input port. The process repeats indefinitely. Please see the flowchart also.




```
;filename: Example_2.asmfa
;Notes:
;       r1 is used as an I/O register
;       r2 is used as a delay counter
;
start:  in r1, 126      ; 126d = 7Eh
        out r1, 222     ; 222d = DEh
;
        movi r2, 0
delay1: subi r2, r2, 1
        jnz r2, [delay1]
;
        movi r1, 0      ; all LEDs off
        out r1, 222
;
        movi r2, 0
delay2: subi r2, r2, 1
        jnz r2, [delay2]
;
        jump [start]
;
        halt
```

It is also possible to use a single address for both the input and the output port. The following diagram shows an address decoder for a 16-bit parallel input/output port at address 2Ch of the FALCON-A's I/O space. Note that the control bus lines IOW# and IOR# will differentiate between the register and the tri-state buffer.

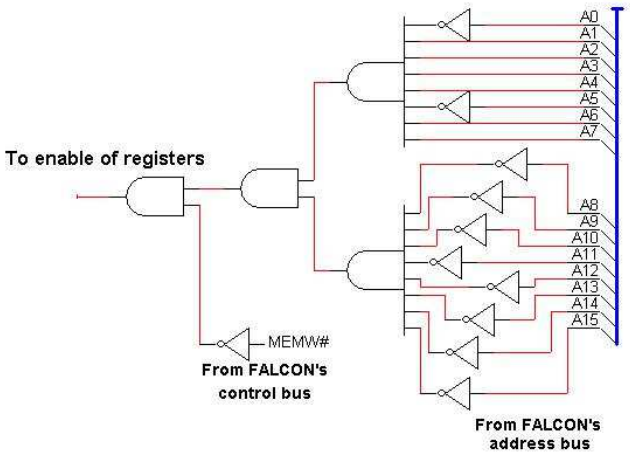


FALCON-A's Address Decoder for an I/O Port at the Address 2Ch

Memory mapped I/O ports

If it is desired to map the 16-bit output port of Example #1(lec24) on the memory space of the FALCON-A, the following changes would be needed.

1. Replace the IOW# signal with the MEMW# signal.
2. Use the entire CPU address bus at the input of the address decoder, as shown in the next figure. This address decoder uses the addresses 00DEh and 00DFh of the FALCON-A's memory space.
3. Use the **store** instruction instead of the **out** instruction for sending data to the output port (for memory mapped input ports, use the **load** instruction instead of the **in** instruction).



Address Decoder for a memory mapped 16-bit parallel output port for the FALCON-A at address 00DEh and 00DFh

The program for Example #2 (lec25) is rewritten for the case of a memory mapped output port, and is shown in the attached text box. The advantage will be that more than 256 ports are available, but the disadvantage is that the address decoder will become more complex, resulting in increased hardware costs.

To avoid the increase in hardware complexity, many architects use what is called "partial decoding". This is explained in the next section.

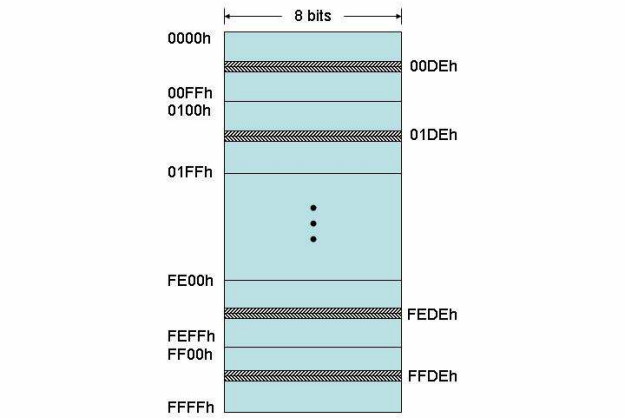
```
;filename: Example_2MM.asmfa
;Notes:
;       For MEMORY MAPPED
;       output port at 00DEh
;
;       r6 holds the output address
;       r7 holds the input address
;
        movi r6, 111
        add r6, r6, r6
;
        movi r7, 126
;
;       r1 is used as an I/O register
;       r2 is used as a delay counter
;
start:  load r1,[r7]    ; 126d = 7Eh
        store r1, [r6]  ; 222d = DEh
;
        movi r2, 0
delay1: subi r2, r2, 1
        jnz r2, [delay1]
;
        movi r1, 0      ; all LEDs off
        store r1, [r6]
;
        movi r2, 0
delay2: subi r2, r2, 1
        jnz r2, [delay2]
;
        jump [start]
;
        halt
```

Partial decoding and the "wrap around" effect

Partial decoding is a technique in which some of the CPU's address lines forming an input to the address decoder are ignored. This reduces the complexity of the address decoder, and also lowers the cost. As an example, if the address lines A8...A15 from the FALCON-A are not used in the address decoder of the previous figure, this will save eight inverters and two AND gates. Partial decoding is an attractive choice in small systems, where the size of the address space is large but most of the memory is unimplemented. However, partial decoding has its price as well. Consider the memory map for the

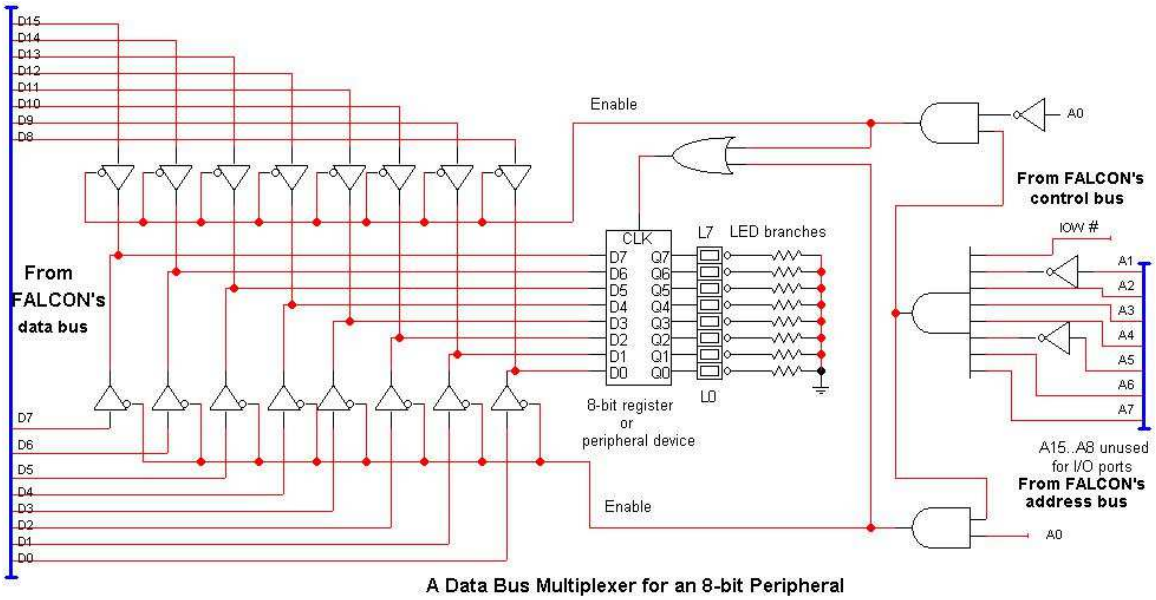
FALCON-A, shown again in the next figure. With 16 address lines, the total address space is 2^{16} = 64 Kbytes. When the upper eight address lines are unused, they become don't cares. The port shown in the previous figure will be accessed for address 00DEh. But, it will also be accessed for address 01DEh, 02DEh... FFDEh. In fact, the 64 Kbyte address space has been reduced to a 256 byte space. It "wrapped around" itself 256 times. If we only left 6 address lines, i.e., A15

... A10, unconnected, then we will still have a "wrap around", but of a different type. Now a 1 Kbyte (= 2^{10}) address area will wrap around itself 64 times (= 2^6).



Data bus multiplexing

Data bus multiplexing refers to the situation when one part of the data bus is connected to the peripheral's data bus at one time and the second part of the data bus is connected to the peripheral's data bus at a different time in such a way that at one time, only one 8-bit portion of the data bus is connected to the peripheral.



A Data Bus Multiplexer for an 8-bit Peripheral using address DCh and DDh.

Consider the situation where an 8-bit peripheral is to be interfaced with a CPU that has a 16-bit (or larger) data bus, but a byte-wide address space. Each byte transferred over the data bus will have a separate address associated with it. For such CPUs, data bus multiplexing can be used to attach 8-bit peripherals requiring a block of addresses. Tri-state buffers can be used for this

purpose as shown in the attached figure. The logic circuit shown is for an 8-bit parallel output port using addresses DCh and DDh of the FALCON's I/O address space. It is assumed that the CPU allows the use of a part of its data bus during a transfer, and that each 16-bit general purpose register can be used as two separate 8-bit registers, e.g., r1 can be split as r1L and r1H such that

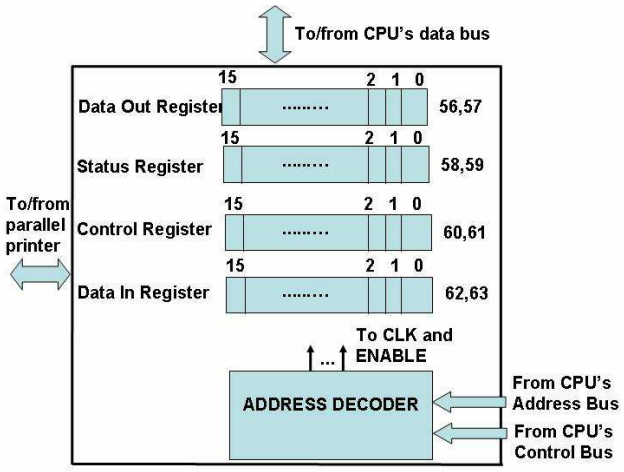
r1L<7..0> := r1<7..0>, and

r1H<7..0> := r1<15..8>

The LED branches and the 8-bit register shown in the diagram serve as a place holder, and can be replaced by a peripheral device in actual practice. For an even address, A0=0, and the upper group of the tri-state buffers is enabled, thereby connecting D<15..8> of the CPU to the peripheral, while for an odd address from the CPU, A0=1, and the lower group of the tri-state buffers is enabled. This causes D<7..0> of the CPU to be connected with the peripheral device. In such systems the instruction **out r1H,220** will access the peripheral device using D<15..8>, while the instruction **out r1L,221** will access it using D<7..0>. The instruction **out r1,220** will

send r1H to the peripheral and the contents of r1L will be lost. Why? This is left as an exercise for the student. The advantage of data bus multiplexing is that all addresses are utilized and none of them is wasted, while the disadvantage is the increased complexity and cost of the interface.

A generic I/O interface



Most parallel I/O ports used with peripheral devices are mapped on a range of contiguous addresses. The following figure shows the block diagram of part of an interface that can be used with a typical parallel printer. It used eight consecutives addresses: address 56 to 63. A similar interface can be used with the FALCON-A. The registers shown within the interface are associated with some parallel device, and have some pre-defined functions. For example, the 16 bit register at addresses 56 and 57 can be used as a "data out" register for sending data bytes to the parallel device. In the same way, the register at addresses 60 and 61 can be used by the CPU to send control bits to the device. The double arrow shown at the top corresponds to the data bus connection of the interface with the CPU. The address decoder shown at the bottom receives address and control information from the CPU and generates enable signals for these registers. These abstract concepts are further explained in Example #3(lec25).

The Centronics Parallel Printer Interface

The Centronics Parallel Printer Interface is an example of a real, industry standard, set of signal specifications used by most printer manufacturers. It was originally developed for Centronics printers and can be used by devices having a uni-directional, byte-wide parallel interface. Table 1 shows the important signals and their functions as defined by the Centronics standard. Note that the direction of the signals is with respect to the printer and not with respect to the CPU.

Typically, the printer (or any other similar device) is connected to the CPU via a cable which has a 25-pin connector at the CPU side and a 36-pin connector at the printer side. Every data bit in the 8-bit data bus D<7…0> uses a twisted pair for suppressing transmission-line effects, like radiation and noise. The return path of these pins should always be connected to signal ground. Additionally, the entire printer cable should be shielded, and connected to chassis ground on each side. The three signals STROBE#, BUSY and ACKNLG# form a set of handshaking signals. By using these signals, the CPU can communicate asynchronously with the printer, as shown in the accompanying timing waveforms. When the printer is ready for printing, the CPU starts data transfer to the printer by placing the 8-bit data (corresponding to the ASCII value of the character to be printed) on the printer's data bus (pin 2 through 9 on the 36-pin connector, as shown in Table 1). After this, a negative pulse of duration at least 0.5µs is applied to the STROBE# input (pin1) of the printer. The minimum set-up and hold times of the latches within the printer are specified as 0.5µs each, and these timing requirements must be observed by the CPU (the interface designer should make sure that these specifications are met). As soon as STROBE# goes low, the printer activates its BUSY line (pin 11) which is an indication to the CPU that additional bytes cannot be accepted. The CPU can monitor this status signal over an input port (a detailed assignment of these signals to I/O port bits is given in Table 2).

Table 1: The Centronics Parallel Printer Interface
(power and ground signals are not shown)

Signal Name	Direction w.r.t. Printer	Function Summary	Pin# (25-DB) CPU side	Pin# (36-DB) Printer side
D<7..0>	Input	8-bit data bus 1-bit control signal	9,8,…,2	9,8,…,2
STROBE#	Input	High: default value. Low: read-in of data is performed. 1-bit status signal	1	1
ACKNLG#	Output	Low: data has been received and the printer is ready to accept new data. High: default value. 1-bit status signal	10	10
BUSY	Output	Low: default value High: see note#1 1-bit status signal	11	11
PE#	Output	High: the printer is out of paper. Low: default value. 1-bit control signal	12	12
INIT#	Input	Low: the printer controller is reset to its initial state and the print buffer is cleared. High: default value. 1-bit status signal	16	31
SLCT	Output	High: the printer is in selected state. 1-bit control signal	13	13
AUTO FEED XT#	Input	Low: paper is automatically fed after one line.	14	14

		1-bit control signal		
SLCT IN#	Input	Low: data entry to the printer is possible. High: data entry to printer is not Possible.	17	36
		1-bit status signal		
ERROR#	Output	Low: see note#2. High: default value.	15	32

Note#1

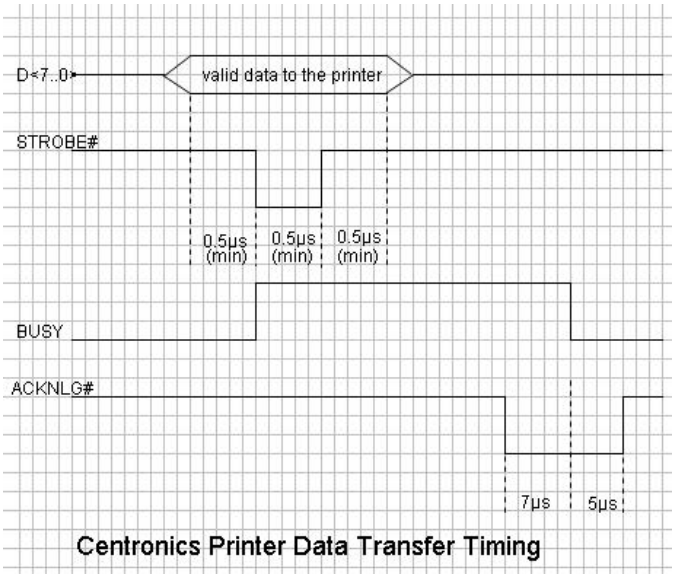
The printer cannot read data due to one of the following reasons:

1. During data entry
2. During data printing
3. In offline state
4. During printer error status

Note#2

When the printer is in one of the following states:

1. Paper end state
2. Offline state
3. Error state



Centronics Printer Data Transfer Timing

When this character is completely received, the ACKNLG# signal (pin 10) goes low, indicating that the transfer is complete. Soon after this, the BUSY signal returns to logic zero, indicating that a new transfer can be initiated. The BUSY signal is more suitable for level-triggered systems, while the ACKNLG# signal is better for edge-triggered systems.

The interface will typically use two eight bit parallel output ports of the CPU, one for the ASCII value of the character byte and the other for the control byte. It also specifies an 8-bit parallel input port for the printer's status information that can be checked by the CPU.

Table 2: Centronics Bit Assignment For I/O Ports

Logical Address	Description	7	6	5	4	3	2	1	0
0	8-bit output port for DATA	D<7>	D<6>	D<5>	D<4>	D<3>	D<2>	D<1>	D<0>
1	8-bit input port for	BUSY	ACKNLG#	PE#	SLCT	ERROR#	Unused	Unused	Unused

	STATUS								
2	8-bit output port for CONTROL	Unused	Unused	DIR[15]	IRQEN	SLCTIN#	INIT#	Auto FeedXT#	STROBE #

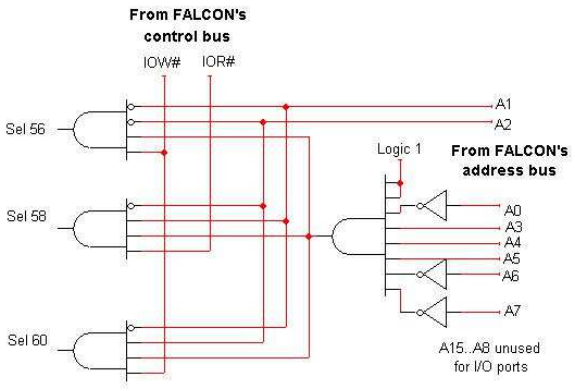
Example # 3:

Problem statement:

Design a Centronics parallel printer interface for the FALCON-A CPU. Map this interface starting at address 38h (56 decimal) of the FALCON-A's I/O address space. Solution:

The Centronics interface requires at least three I/O addresses. However, since the FALCON-A has a 16-bit data bus, and since we do not want to implement data bus multiplexing (to keep things simple), we will use three contiguous even addresses, i.e., 38h, 3Ah and 3Ch for the address decoder design. This arrangement also conforms to the requirements of our assembler.

Moreover, we will connect data bus lines D7...D0 of the FALCON-A to the 8-bit data bus of the printer (i.e. pins 9, 8, ... , 2 of the printer cable) and leave lines D15...D8 unconnected. Since the FALCON-A uses the big-endian format, this will make sure that the low byte of CPU registers will be transferred to the printer. (Recall that these bytes will actually be mapped on addresses 39h, 3Bh and 3Dh). The logic diagram of the address decoder for this interface is shown in the given figure.



Address decoder for three parallel ports for the FALCON-A at addresses 38h, 3Ah, and 3Ch

[15] This bit, when set, enables the bidirectional mode.

Lecture No. 26

Programmed I/O

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.2.2

Summary

- The Centronic Parallel Printer Interface(Cont.)
- Programmed Input/Output
- Examples of Programmed I/O for FALCON-A and SRC
- Comparisons of FALCON-A, SRC examples

The Centronic Parallel Printer Interface (Cont.)

Table 1: The Centronics Parallel Printer Interface (power and ground signals are not shown)
(The explanation of this table is provided in lecture 25 also)

Signal Name	Direction w.r.t. Printer	Function Summary	Pin# (25-DB) CPU side	Pin# (36-DB) Printer side
D<7..0>	Input	8-bit data bus 1-bit control signal	9,8,…,2	9,8,…,2
STROBE#	Input	High: default value. Low: read-in of data is performed. 1-bit status signal	1	1
ACKNLG#	Output	Low: data has been received and the printer is ready to accept new data. High: default value. 1-bit status signal	10	10
BUSY	Output	Low: default value High: see note#1 1-bit status signal	11	11

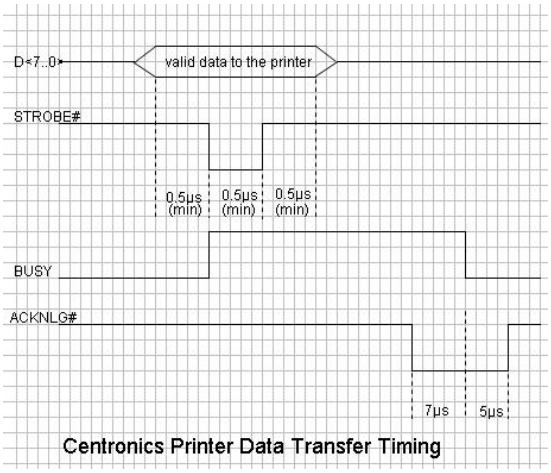
PE#	Output	High: the printer is out of paper. Low: default value. 1-bit control signal	12	12
INIT#	Input	Low: the printer controller is reset to its initial state and the print buffer is cleared. High: default value. 1-bit status signal	16	31
SLCT	Output	High: the printer is in selected state. 1-bit control signal	13	13
AUTO FEED XT#	Input	Low: paper is automatically fed after one line. 1-bit control signal	14	14
SLCT IN#	Input	Low: data entry to the printer is possible. High: data entry to printer is not Possible. 1-bit status signal	17	36
ERROR#	Output	Low: see note#2. High: default value.	15	32

Note#1
The printer cannot read data due to one of the following reasons:

5. During data entry
6. During data printing
7. In offline state
8. During printer error status

Note#2
When the printer is in one of the following states:

4. Paper end state
5. Offline state
6. Error state



Centronics Printer Data Transfer Timing

When this character is completely received, the ACKNLG# signal (pin 10) goes low, indicating that the transfer is complete. Soon after this, the BUSY signal returns to logic zero, indicating that a new transfer can be initiated. The BUSY signal is more suitable for level-triggered systems, while the ACKNLG# signal is better for edge-triggered systems.

The interface will typically use two eight bit parallel output ports of the CPU, one for the ASCII value of the character byte and the other for the control byte. It also specifies an 8-bit parallel input port for the printer's status information that can be checked by the CPU.

Table 2: Centronics Bit Assignment For I/O Ports

Logical Address	Description	7	6	5	4	3	2	1	0
0	8-bit output port for DATA	D<7>	D<6>	D<5>	D<4>	D<3>	D<2>	D<1>	D<0>
1	8-bit input port for STATUS	BUSY	ACKNLG#	PE#	SLCT	ERROR#	Unused	Unused	Unused
2	8-bit output port for CONTROL	Unused	Unused	DIR[15]	IRQEN	SLCTIN#	INIT#	AutoFeedXT#	STROBE #

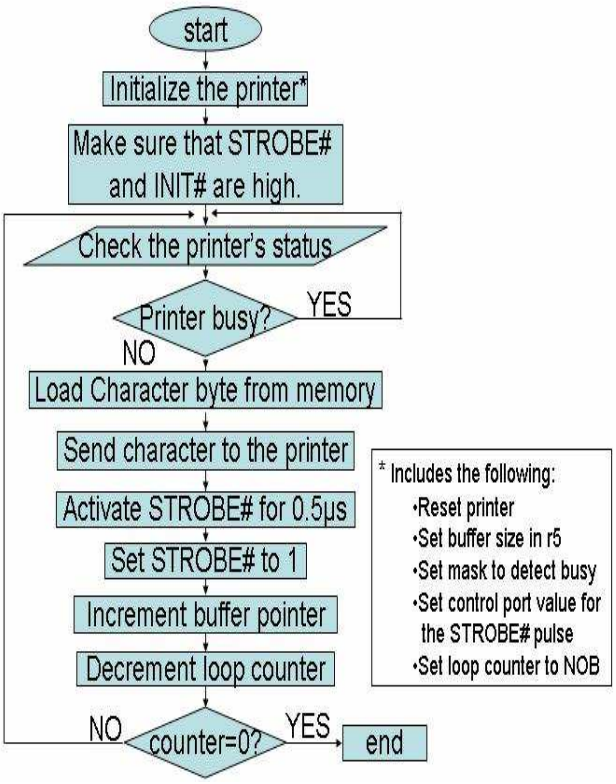
Example # 1

Problem statement:

Assuming that a Centronics parallel printer is interfaced to the FALCON-A processor, as shown in example 3 of lecture 25, write an assembly language program to send an 80 character line to the printer. Assume that the line of characters is stored in the memory starting at address 1024.

Solution:

The flowchart for the solution is shown in given figure and the program listing is shown in the textbox with filename: Example_1.



The first thing that needs to be done is the initialization of the printer. This means that a "reset" command should be sent to the printer. Using the information from Table 1, this can be done by writing a 0 to bit 2 (i.e., INIT#) of the control register having logical address 2. In our example, this maps onto address 60 of the FALCON-A. (Remember to set this bit to logic 1 for normal operation of the printer). Then we make STROBE# high by placing logic 1 in bit 0 of the control register. Bit 1 and bit 3 should be 0 because we want to activate auto line feed and keep the printer in selected mode. Additionally, bit 4 and bit 5 should be 0 so that interrupts are disabled and the bi-directional mode is not selected. The complete control word is 0000 0001 and this value has been assigned to the variable reset in the program. The following instruction pair performs the reset operation:

```
movi r1, reset
out r1, controlp
```

As it is given that the starting address of the printer buffer is 1024[17], so we place this address in r5. The mask to test the BUSY flag is placed in r3. The value for the mask is 80h. This corresponds to a logic 1 in bit 7 and logic zeros elsewhere for the status register having address 58 (logical address 1 in Table 1). Then the program enters a loop, called the polling loop, to test the status of the printer. If the printer is busy, the loop repeats. The following three instructions form the polling loop:

```
in r1, statusp
and r1, r1, r3
jnz r1, [again]
```

The status of the printer is placed in register r1, and bit 7 is tested for logic 0. If not so, the program repeats the status check operation.

When the printer is ready to accept a new character, it clears bit 7 (i.e., the BUSY bit) of the status register. At this time, the program picks the next character from the memory and sends it to the printer. The STROBE# line is activated and then it is deactivated to generate the necessary pulse on this input of the printer. Finally, the buffer pointer is advanced, the loop counter is decremented and the process repeats. When all the characters have been printed, the program halts.

A number of equates have been used in the program to make it flexible as well as easily readable. The program is shown on the next page.

---

[17] The **mul** instruction is used for this purpose because the 8-bit immediate operand in the **movi** instruction can only be within the range –128 and +127. Using the **mul** instruction in this way overcomes the limitation of the FALCON-A. Similarly, the **shiftl** instruction is used to bring 80h in register r3.

```
; filename: Example_1.asmfa
;
; This program sends an 80 character line
; to a FALCON-A parallel printer
;
; Notes:
; 1.  8-bit printer data bus connected to
;     D<7...0> of the FALCON-A (remember big-endian)
;     Thus, the printer actually uses addresses 57, 59 & 61
;
; 2.  one character per 16-bits of data xfered
;
;
      .org 400
;
NOB:        .equ  80
;
      movi r5, 32
      mul r5, r5, r5    ; r5 holds 1024 temporarily
;
      movi r3, 1
      shiftl r3, r3, 7  ; to set mask to 0080h
;
datap:      .equ 56
statusp:    .equ 58
controlp:   .equ 60
;
reset:            .equ 1
; used to set unidirectional, no interrupts,
; auto line feed, and strobe high
;
strb_H:           .equ 5
strb_L:           .equ 4
;
      movi r1 reset     ; use r1 for data xfer
      out r1, controlp
;
      movi r7, NOB      ; use r7 as character counter
;

again:      in r1, statusp
;
      and r1, r1, r3    ; test if BUSY = 1?
      jnz r1, [again]   ; wait if BUSY = 1
;
      load r1, [r5]
      out r1, datap
      movi r1, strb_L
      out r1, controlp
      movi r1, strb_H
      out r1, controlp
      addi r5, r5, 2
      subi r7, r7, 1
      jnz r7, [again]
      halt
```

I/O techniques:

There are three main techniques using which a CPU can exchange data with a peripheral device, namely

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA).

In this section, we present the first one.

Programmed Input/Output

Programmed I/O refers to the situation when all I/O operations are performed under the direct control of a program running on the CPU. This program, which usually consists of a "tight loop", controls all I/O activity, including device status sensing, issuing read or write commands, and transferring the data[18]. A subsequent I/O operation cannot begin until the current I/O operation to a certain device is complete. This causes the CPU to wait, and thus makes the scheme extremely inefficient. The solution to Example # 3(lec24), Example #2(lec25), and Example #1(lec26) are examples of programmed input/output. We will analyze the program for Example #1(lec26) to explain a few things related to the programmed I/O technique.

Timing analysis of the program in Example # 1(lec26)

The main loop of the program given in the solution to Example #1(lec26) executes 80 times. This is equal to the number of characters to be printed on one line. This portion of the program is shown again with the execution time of each instruction listed in brackets with it. The numbers shown are for a uni-bus CPU implementation. A complete list of execution times for all the FALCON-A's instructions is given in Appendix A. A number of things can be noted now.

```
        movi r7, NOB          [2]
;
again:  in r1, statusp        [3]
        and r1 , r1, r3       [3]
        jnz  r1, [again]      [4]
;
        load r1, [r5]         [5]
        out r1, datap         [3]
        movi r1, strob_L      [2]
        out r1, controlp      [3]
        movi r1, s trob_H     [2]
        out r1, controlp      [3]
        addi r5, r5, 2        [3]
        subi r7, r7, 1        [3]
        jnz r7, [again]       [4]
        halt
```

1. Assuming that the output at the hardware pins changes at the end of the (I/O write) bus cycle, the STROBE# signal will go from logic1 to logic 0 at the end of the instruction pair.

```
movi r1, strb_L [2]
out r1, controlp [3]
```

The execution time for these two instructions is 2+3 = 5 clock periods. Therefore, STROBE# stays at logic1 for at least 5 clock periods i.e., during these two instructions. For a 10MHz FALCON-A CPU, this will correspond to 5x100 = 500nsec = 0.5lsec.

[18] The I/O device has no direct access to the memory or the CPU, and transfer is generally done by using the CPU registers.

Since the data to the printer is being sent by the CPU using the two instructions (**load r1, [r5]** and **out r1, datap**) which are before the first **movi** instruction, the printer's data setup time requirement is satisfied as long as we do not increase the clock frequency beyond 10MHz.

After these two instructions, the next two instructions in the program cause STROBE# to go to logic 1 again.

```
movi r1, strb_H      [2]
out r1, controlp     [3]
```

These two instructions also take 5 clock periods, or 0.5lsec, to execute. Thus, the timing requirement of the STROBE# pulse width will also be satisfied as long as we do not increase the clock frequency beyond 10MHz. In case the frequency is greater than 10MHz, other instruction can be used in between these two pairs of instructions.

The printer's data hold time requirement is easily satisfied because there are a number of instructions after this **out** instruction which do not change the control port, and the character value is already present in the data register within the interface since the end of the **out r1, datap** instruction.

2. The three instructions given below:

```
again:  in r1, statusp [3]
        and r1, r1, r3 [3]
        jnz r1, [again] [4]
```

form what is called a "polling loop". The process of periodically checking the status of a device to see if it is ready for the next I/O operation is called "polling". It is the simplest way for an I/O device to communicate with the CPU. The device indicates its readiness by setting certain bits in a status register, and the CPU can read these bits to get information about the device. Thus, the CPU does all the work and controls all the I/O activities. The polling loop given above takes 10 clock periods. For a 10MHz FALCON-A CPU, this is 10x100=1lsec. One pass of the main loop takes a total of 3+3+4+5+3+2+3+2+3+3+3+4 = 38 clock periods which is 38x100 = 3.8lsec. This is the time that the CPU takes to send one character to the printer. If we assume that a 1000 character per second (cps) printer is connected to the CPU, then this printer has the capability to print one character in every 1msec or every 1000lsec. So, after sending a character in 3.8lsec to the printer, the CPU will wait for about 996lsec before it can send the next character to the printer. This implies that the polling loop will be executed about 996 times for each character. This is indeed a very inefficient way of sending characters to the printer.

An improved way of doing this would be to include a memory of suitable size within the printer. This memory is also called a buffer, as explained earlier. The CPU can fill this buffer in a single "burst" at its own speed, and then do something else, while the printer picks up one character at a time from this buffer and prints it at its own speed. This is exactly the situation with today's printers. The task of generating the STROBE# pulse will also be done by the electronic circuits within the printer. In effect, a dedicated processor within the printer will do this job. However, if the buffer within the printer fills up, the CPU will still not be able to transfer additional data to it. A different handshaking scheme will then be needed to make the CPU to communicate asynchronously with the buffer in the printer, resulting in an inefficient operation again. This is explained below.

Assume that the printer has a FIFO type buffer of size 64 bytes that can be filled up without any delay at the time when the printer is not printing anything. When one or more character values are present in the buffer, the printer will pick up one value at a time and print it. Remember we have a 1000 cps printer, so it takes 1msec to print a character. The program for Example #1(lec26) is modified for this situation and is given below. All the assumptions are the same, unless otherwise mentioned.

```
again:  in r1, statusp        [3]
        and r1, r1, r3        [3]
        jnz r1, [again] [4]
        load r1, [r5]         [5]
        out r1, datap         [3]
        addi r5, r5, 2        [3]
        subi r7, r7, 1        [3]
        jnz r7, [again] [4]
```

Note that while the instructions for generating the STROBE# pulse have been eliminated, the polling loop is still there. This is necessary because the BUSY signal will still be present, although it will have a different meaning n now. In this case, BUSY =1 will mean that the buffer within the printer is full and it cannot accept additional bytes.

The main loop shown in the program has an execution time of 28 clock periods, which is 2.8lsec for a 10MHz FALCON-A CPU. The polling loop still takes 10 clock periods or 1lsec. Assuming that this program starts when the buffer in the printer is empty, the outer loop will execute 64 times before the CPU encounters a BUSY=1 condition. After that the situation will be the same as in the previous case. The polling loop will execute for about 996 times before BUSY goes to logic 0. This situation will persist for the remaining 16 characters (remember we are sending an 80 character line to the printer).

One can argue that the problem can be solved by increasing the buffer size to more than 80 bytes. Well, first of all, memory is not free. So, a large buffer will increase the cost of the printer. Even if we are willing to pay more for an improved printer, the larger buffer will still fill up whenever the number of characters is more than the buffer size. When that happens, we will be back to square one again.

A careful analysis of the situation reveals that there is something wrong with the scheme that is being used to send data to the printer. This problem of having a larger overhead of polling was recognized long ago, and therefore, interrupts were invented as an alternate to programmed I/O. Interrupt driven I/O will be the topic of the next lecture.

Programmed I/O in SRC

In this section, we will discuss some more examples of programmed I/O with our example processor SRC which uses the memory mapped I/O technique.

Program for Character Output

To understand how programmed I/O works in SRC, we will discuss a program which outputs the character to the printer. The first instruction loads the branch target and the second instruction loads the character into lower 8 bits of register r2. The 2-instruction loop reads the status register and tests the ready signal by checking its sign bit. It executes until the ready signal becomes logic one. On exit from the loop, the character is written to the device data register by the store instruction.

```
        lar r3, wait
        ldr r2, char
wait:   ld r1, COSTAT
        brpl r3, r1
        st r2, COUT
```

A 10 MIPS, SRC would execute 10,000 instructions waiting for a 1,000 character/sec printer.

Program Fragment to Print 80-Character Line

The next example for the SRC is of a program which sends an 80-character line to a line printer with a command register. There are two nested loops starting at label wait. The two instruction inner loop, which waits for ready and the outer seven instruction loop which performs the following tasks.

- Outputs a character
- Advance the buffer pointer
- Decrement the register containing the number of characters left to print
- Repeat if there are more characters left to send.

The last two instructions issue the command to print the line.

The next example discussed from the book is of a driver program for 32-character input devices (Figure 8.10, Page 388).

388 Chapter 8 Input and Output

FIGURE 8.10 Programmed I/O Driver for 32 Character Input Devices

```
;Driver for 32 char. input devices. Register usage:
;r0 - working register. r1 - stores a char. into
;selected buffer. r2 - indexes (CIN, CICTL) pairs
;of I/O registers & (Bufp, Done) pairs controlling
;input lines. r3 - =1 at the end of a pass only if
;no device is active.
CICTL    .equ    FFFFF300H     ;First char.-in control reg.
CIN      .equ    FFFFF304H     ;First input data register.
CR       .equ    13            ;ASCII carriage return.
Bufp:    .dcw    1             ;First pointer into a buffer.
Done:    .dcw    63            ;1st done & rest of pointers.
Driver:  lar     r4, Next      ;Branch targets: move to next
         lar     r5, Check     ;  char., check device active,
         lar     r6, Start     ;  & start a new polling pass.
Start:   la      r2, 0         ;Point to first device, &
         la      r3, 1         ;  set all inactive flag.
Check:   ld      r0, Done(r2)  ;If device not still active,
         brmi    r4, r0        ;  go advance to next.
         ld      r3, 0         ;Clear the all inactive flag.
         ld      r0, CICTL(r2) ;Get device ready flag, & go
         brpl    r4, r0        ;  move to next if not ready.
         ld      r0, CIN(r2)   ;Get character and
         ld      r1, Bufp(r2)  ;  correct buffer pointer, &
         st      r0, 0(r1)     ;  store character in buffer.
         addi    r1, r1, 4     ;Advance character pointer,
         st      r1, Bufp(r2)  ;  and return it to memory.
         addi    r0, r0,-CR    ;If not carriage return,
         brnz    r4, r0        ;  go advance to next device.
         la      r0, -1        ;Set done flag to -1 on
         st      r0, Done(r2)  ;  detecting carriage return.
Next:    addi    r2, r2,8      ;Advance device pointer, and
         addi    r0, r2,-256   ;  if not last device,
         brnz    r5, r0        ;  go check next one.
         brzr    r6, r3        ;If any active, make new pass.
```

Comparisons of the SRC and FALCON-A Examples

The FALCON-A and SRC programmed I/O examples discussed are similar with some differences. In the first example discussed for the SRC (i.e. Character output), the control signal responsible for data transfer by the CPU is the ready signal while for FALCON-A Busy (active low)signal is checked. In the second example for the SRC, the instruction set, address width and no. of lines on address is different.

Although different techniques have been used to increase the efficiency of the programmed I/O, overheads due to polling cannot be completely eliminated.

Lecture No. 27

Interrupt Driven I/O

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.2.2

Summary

- Programmed I/O Driver for SRC
- Interrupt Driven I/O

Programmed I/O Driver for SRC

Please refer to Figure 8.10 of the text and its associated explanation.

388 Chapter 8 Input and Output

FIGURE 8.10 Programmed I/O Driver for 32 Character Input Devices

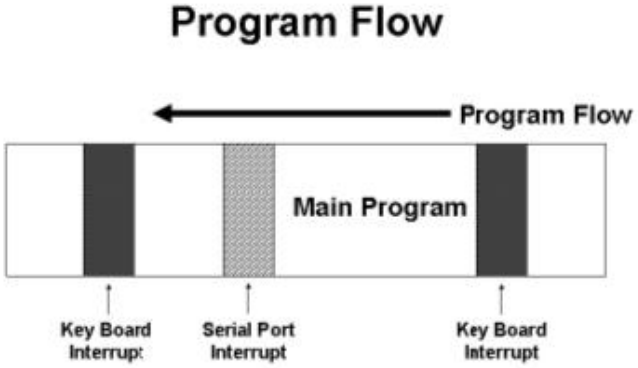
```
;Driver for 32 char. input devices. Register usage:
;r0 - working register.  r1 - stores a char. into
;selected buffer.  r2 - indexes (CIN, CICTL) pairs
;of I/O registers & (Bufp, Done) pairs controlling
;input lines.  r3 - =1 at the end of a pass only if
;no device is active.

CICTL    .equ    FFFFF300H     ;First char.-in control reg.
CIN      .equ    FFFFF304H     ;First input data register.
CR       .equ    13            ;ASCII carriage return.
Bufp:    .dcw    1             ;First pointer into a buffer.
Done:    .dcw    63            ;1st done & rest of pointers.
Driver:  lar     r4, Next      ;Branch targets: move to next
         lar     r5, Check     ;   char., check device active,
         lar     r6, Start     ;   & start a new polling pass.
Start:   la      r2, 0         ;Point to first device, &
         la      r3, 1         ;   set all inactive flag.
Check:   ld      r0, Done(r2)  ;If device not still active,
         brmi    r4, r0        ;   go advance to next.
         ld      r3, 0         ;Clear the all inactive flag.
         ld      r0, CICTL(r2) ;Get device ready flag, & go
         brpl    r4, r0        ;   move to next if not ready.
         ld      r0, CIN(r2)   ;Get character and
         ld      r1, Bufp(r2)  ;   correct buffer pointer, &
         st      r0, 0(r1)     ;   store character in buffer.
         addi    r1, r1, 4     ;Advance character pointer,
         st      r1, Bufp(r2)  ;   and return it to memory.
         addi    r0, r0,-CR    ;If not carriage return,
         brnz    r4, r0        ;   go advance to next device.
         la      r0, -1        ;Set done flag to -1 on
         st      r0, Done(r2)  ;   detecting carriage return.
Next:    addi    r2, r2,8      ;Advance device pointer, and
         addi    r0, r2,-256   ;   if not last device,
         brnz    r5, r0        ;   go check next one.
         brzr    r6, r3        ;If any active, make new pass.
```

Interrupt Driven I/O:

Introduction:

An interrupt is a request to the CPU to suspend normal processing and temporarily divert the flow of control through a new program. This new program to which control is transferred is called an Interrupt Service Routine or ISR. Another name for an ISR is an Interrupt Handler.



- Interrupts are used to demand attention from the CPU.
- Interrupts are asynchronous breaks in program flow that occur as a result of events outside the running program.
- Interrupts are usually hardware related, stemming from events such as a key or button press, timer expiration, or completion of a data transfer.

The basic purpose of interrupts is to divert CPU processing only when it is required. As an example let us consider the example of a user typing a document on word-processing software running on a multi-tasking operating system. It is up to the software to display a character when the user presses a key on the keyboard. To fulfill this responsibility the processor can repeatedly poll the keyboard to check if the user has pressed a key. However, the average user can type at most 50 to 60 words in a minute. The rate of input is much slower than the speed of the processor. Hence, most of the polling messages that the processor sends to the keyboard will be wasted. A significant fraction of the processor's cycles will be wasted checking for user input on the keyboard. It should also be kept in mind that there are usually multiple peripheral devices such as mouse, camera, LAN card, modem, etc. If the processor would poll each and every one of these devices for input, it would be wasting a large amount of its time. To solve this problem, interrupts are integrated into the system. Whenever a peripheral device has data to be exchanged with the processor, it interrupts the processor; the processor saves its state and then executes an interrupt handler routine (which basically exchanges data with the device). After this exchange is completed, the processor resumes its task. Coming back to the keyboard example, if it takes the average user approximately 500 ms to press consecutive keys a modern processor like the Pentium can execute up to 300,000,000 instructions in these 500 Ms. Hence, interrupts are an efficient way to handle I/O compared to polling.

Advantages of interrupts:

- Useful for interfacing I/O devices with low data transfer rates.
- CPU is not tied up in a tight loop for polling the I/O device.

Program Flow for an interrupt driven interface:

The attached figure shows the program flow executing on a processor with interrupts enabled. As we can see, the program is interrupted in several locations to service various types of interrupts.

Types of Interrupts:

The general categories of interrupts are as follows:

- Internal Interrupts
- External Interrupts
- Hardware Interrupts
- Software Interrupts

Internal Interrupts:

- Internal interrupts are generated by the processor.
- These are used by processor to handle the exceptions generated during instruction execution.

Internal interrupts are generated to handle conditions such as stack overflow or a divide-by-zero exception. Internal interrupts are also referred to as traps. They are mostly used for exception handling. These types of interrupts are also called exceptions and were discussed previously.

External Interrupts:

External interrupts are generated by the devices other than the processor. They are of two types.

- Hardware interrupts are generated by the external hardware.
- Software interrupts are generated by the software using some interrupt instruction.

As the name implies, external interrupts are generated by devices external to the CPU, such as the click of a mouse or pressing a key on a keyboard. In most cases, input from external sources requires immediate attention. These events require a quick service by the software, e.g., a word processing software must quickly display on the monitor, the character typed by the user on the keyboard. A mouse click should produce immediate results. Data received from the LAN card or

the modem must be copied from the buffer immediately so that pending data is not lost because of buffer overflow, etc.

Hardware interrupts:

Hardware interrupts are generated by external events specific to peripheral devices. Most processors have at least one line dedicated to interrupt requests. When a device signals on this specific line, the processor halts its activity and executes an interrupt service routine. Such interrupts are always asynchronous with respect to instruction execution, and are not associated with any particular instruction. They do not prevent instruction completion as exceptions like an arithmetic overflows does. Thus, the control unit only needs to check for such interrupts at the start of every new instruction. Additionally, the CPU needs to know the identification and priority of the device sending the interrupt request.

There are two types of hardware interrupt:
Maskable Interrupts
Non-maskable Interrupts

Maskable Interrupts:

- These interrupts are applied to the INTR pin of the processor.
- These can be blocked by resetting the flag bit for the interrupts.

Non-maskable Interrupts:

- These interrupts are detected using the NMI pin of the processor.
- These can not be blocked or masked.
- Reserved for catastrophic event in the system.

Software interrupts:

Software interrupts are usually associated with the software. A simple output operation in a multitasking system requires software interrupts to be generated so that the processor may temporarily halt its activity and place the data on its data bus for the peripheral device. Output is usually handled by interrupts so that it appears interactive and asynchronous. Notification of other events, such as expiry of a software timer is also handled by software interrupts. Software

interrupts are also used with system calls. When the operating system switches from user mode to supervisor mode it does so through software interrupts. Let us consider an example where a user program must delete a file. The user program will be executing in the user mode. When it makes the specific system call to delete the file, a software interrupt will be generated, this will cause the processor to halt its current activity (which would be the user program) and switch to supervisor mode. Once in supervisor mode, the operating system will delete the file and then control will return to the user program. While in supervisor mode the operating system would need to decide if it could delete the specified file without harmful consequences to the systems integrity, hence it is important that the system switch to supervisor mode at each system call.

I/O Software System Layers:
The above diagram shows the various software layers related to I/O. At the bottom lies the actual hardware itself, i.e. the peripheral device. The peripheral device uses the hardware interrupts to communicate with the processor. The processor responds by executing the interrupt handler for that particular device. The device drivers form the bridge between the hardware and the software. The operating system uses the device drivers to communicate with the device in a hardware independent fashion, e.g., the operating system need not cater for a specific brand of CRT monitors, or keyboards, the specific device driver written for that monitor or keyboard will act as an intermediary between the operating system and the device. It would be clear from the previous statement that the operating system expects certain common functions from all brands of devices in a category. Actually implementing these functions for each particular brand or vendor is the responsibility of the device driver. The user programs run at top of the operating system.

Interrupt Service Routine (ISR):
- It is a routine which is executed when an interrupt occurs.
- Also known as an Interrupt Handler.
- Deals with low-level events in the hardware of a computer system, like a tick of a real-time clock.

As it was mentioned earlier, an interrupt once generated must be serviced through an interrupt service routine. These routines are stored in the system memory ready for execution. Once the interrupt is generated, the processor must branch to the location of the appropriate service routine to execute it. The branch address of the ISR is discussed next.

Branch Address of the ISR:
There are two ways used to choose the branch address of an Interrupt Service Routine.
Non-vectored Interrupts Vectored Interrupts

Non-vectored Interrupts:
In non-vectored interrupts, the branch address of the interrupt service routine is fixed. The code for the ISR is loaded at fixed memory location. Non-vectored interrupts are very easy to implement and not flexible at all. In this case, the number of peripheral devices is fixed and may not be increased. Once the interrupt is generated the processor queries each peripheral device to find out which device generated the interrupt. This approach is the least flexible for software interrupt handling.

Vectored Interrupts:
Interrupt vectors are used to specify the address of the interrupt service routine. The code for ISR can be loaded anywhere in the memory. This approach is much more flexible as the programmer may easily locate the interrupt vector and change its addresses to use custom interrupt servicing routines. Using vectored interrupts, multiple devices may share the same interrupt input line to the processor. A process called daisy chaining is then used to locate the interrupting device.

Interrupt Vector:
Interrupt vector is a fixed size structure that stores the address of the first instruction of the ISR.
Interrupt Vector Table:
- All of the interrupt vectors are stored in the memory in a special table called Interrupt Vector Table.
- Interrupt Vector Table is loaded at the memory location 0 for the 8086/8088.

Interrupts in Intel 8086/8088:

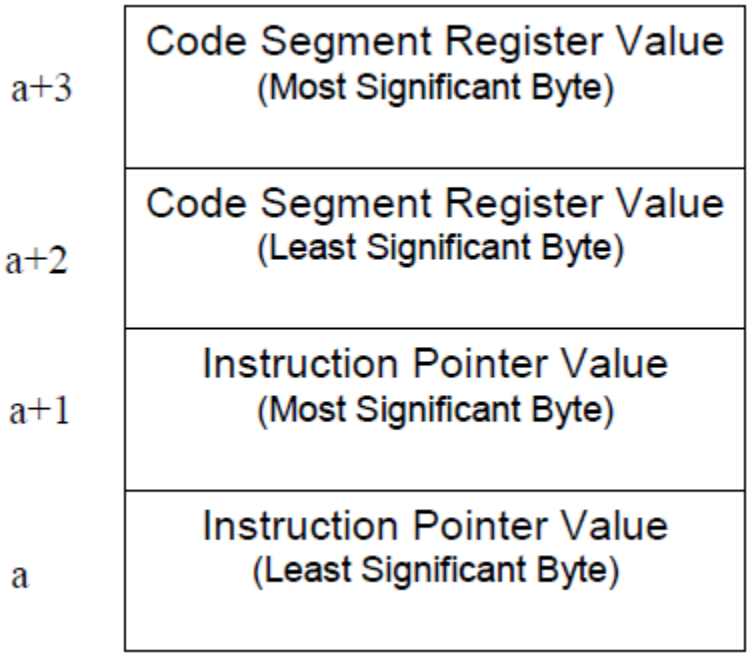
- Interrupts in 8086/8088 are vector interrupts.
- Interrupt vector is of 4 bytes to store IP and CS.
- Interrupt vector table is loaded at address 0 of main memory.
- There is provision of 256 interrupts.

Branch Address Calculation:

- The number of interrupt is the number of interrupt vector in the interrupt vector table.
- Since size of each vector is 4 bytes and interrupt vector starts from address 0, therefore, the address of interrupt vector can be calculated by simply multiplying the number by 4.

Interrupt Vector Example:

In 8086/8088 machines the size of interrupt vector is 4 bytes that holds IP and CS of ISR.



Returning from the ISR:

Every ISA should have an instruction, like the **IRET** instruction, which should be executed when the ISR terminates. This means that the **IRET** instruction should be the last instruction of every ISR. This is, in effect, a FAR RETURN in that it restores a number of registers, and flags to their value before the ISR was called. Thus the previous environment is restored after the servicing of the interrupt is completed.

Interrupt Handling:

The CPU responds to the interrupt request by completing the current instruction, and then storing the return address from PC into a memory stack. Then the CPU branches to the ISR that processes the requested operation of data transfer. In general, the following sequence takes place.

Hardware Interrupt Handling:

- Hardware issues interrupt signal to the CPU.
- CPU completes the execution of current instruction. CPU acknowledges interrupt.
- Hardware places the interrupt number on the data bus.
- CPU determines the address of ISR from the interrupt number available on the data bus.

- CPU pushes the program status word (flags) on the stack along with the current value of program counter.
- The CPU starts executing the ISR.
- After completion of the ISR, the environment is restored; control is transferred back to the main program.

Interrupt Latency:
Interrupt Latency is the time needed by the CPU to recognize (not service) an interrupt request. It consists of the time to perform the following:

- Finish executing the current instruction.
- Perform interrupt-acknowledge bus cycles.
- Temporarily save the current environment.
- Calculate the IVT address and transfer control to the ISR.

If wait states are inserted by either some memory module or the device supplying the interrupt type number, the interrupt latency will increase accordingly.

Interrupt Latency for external interrupts depends on how many clock periods remain in the execution of the current instruction.

On the average, the longest latency occurs when a multiplication, division or a variable-bit shift or rotate instruction is executing when the interrupt request arrives.

Response Deadline:
It is the maximum time that an interrupt handler can take between the time when interrupt was requested and when the device must be serviced.

Expanding Interrupt Structure:
When there is more than one device that can interrupt the CPU, an Interrupt Controller is used to handle the priority of requests generated by the devices simultaneously.

Interrupt Precedence:
Interrupts occurring at the same time i.e. within the same instruction are serviced according to a pre-defined priority.

- In general, all internal interrupts have priority over all external interrupts; the single-step interrupt is an exception.
- **NMI** has priority over **INTR** if both occur simultaneously.
- The above mentioned priority structure is applicable as far as the recognition of (simultaneous) interrupts is concerned. As far as servicing (execution of the related ISR) is concerned, the single-step interrupt always gets the highest priority, then the **NMI**, and finally those (hardware or software) interrupts that occur last. If IF is not 1, then **INTR** is ignored in any case. Moreover, since any ISR will clear **IF**, **INTR** has lower "service priority" compared to software interrupts, unless the ISR itself sets **IF**=1.

Simultaneous Hardware Interrupt Requests:
The priority of the devices requesting service at the same time is resolved by using two ways:
Daisy-Chained Interrupt Parallel Priority Interrupt

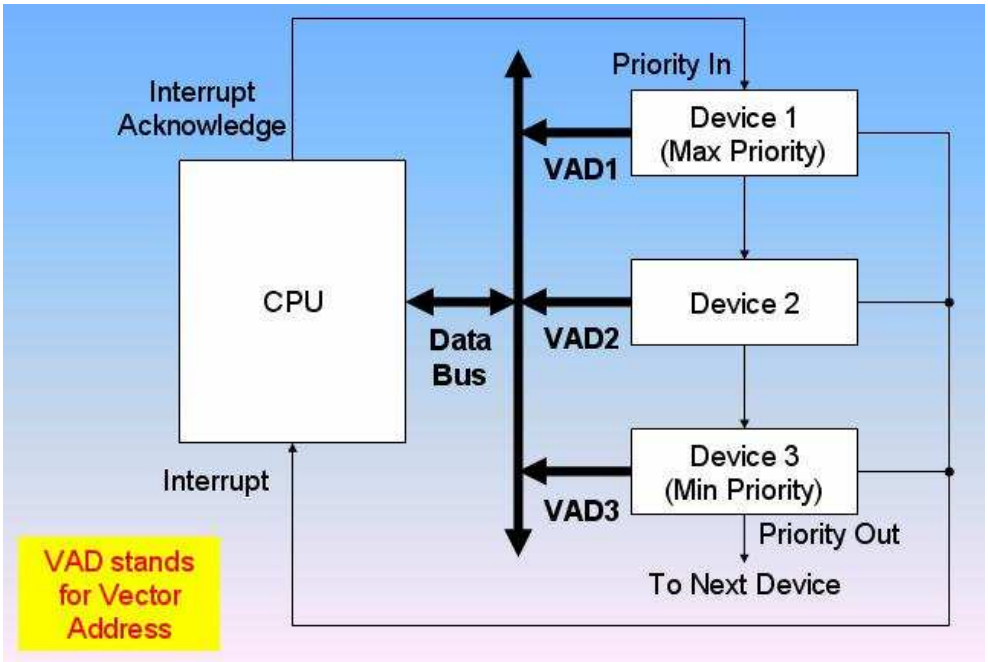
Daisy-Chaining Priority:

- The daisy-chaining method to resolve the priority consists of a series connection of the devices in order of their priority.
- Device with maximum priority is placed first and device with least priority is placed at the end.

Daisy-Chain Priority Interrupt

- The devices interrupt the CPU.
- The CPU sends acknowledgement to the maximum priority device.
- If the interrupt was generated by the device, the interrupt for the device is serviced.
- Otherwise the acknowledgement is passed to the next device.

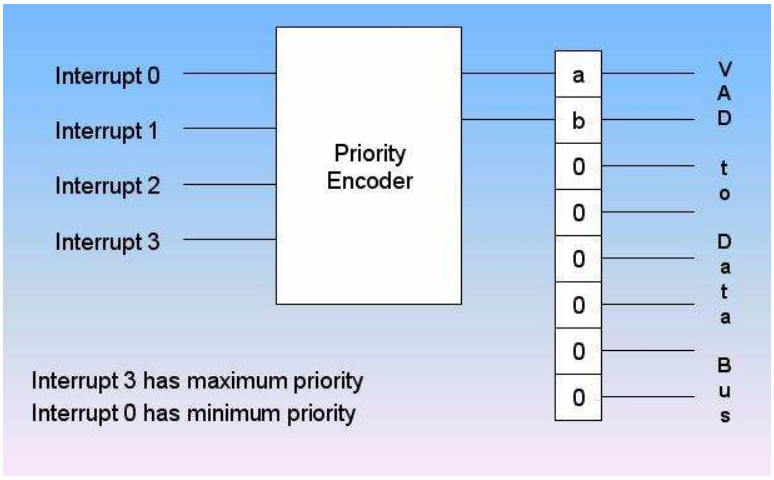
If the higher priority devices are going to interrupt continuously then the device with the lower priority is not serviced. So some additional circuitry is also needed to introduce fairness.



Parallel Priority:

- Parallel priority method for resolving the priority uses individual bits of a priority encoder.
- The priority of the device is determined by position of the input of the encoder used for the interrupt.

Parallel Priority Interrupt:


Interrupt 0
Interrupt 1
Interrupt 2
Interrupt 3
Priority Encoder
a
b
0
0
0
0
0
0
VAD to Data Bus
Interrupt 3 has maximum priority
Interrupt 0 has minimum priority

Lecture No. 28

Interrupt Hardware and Software

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.3

Summary

- Comparison of Interrupt driven I/O and Polling
- Design Issues
- Interrupt Handler Software
- Interrupt Hardware
- Interrupt Software

Comparison of Interrupt driven I/O and Polling

Interrupt driven I/O is better than polling. In the case of polling a lot of time is wasted in questioning the peripheral device whether it is ready for delivering the data or not. In the case of interrupt driven I/O the CPU time in polling is saved.

Now the design issues involved in implementation of the interrupts are twofold. There would be a number of interrupts that could be initiated. Once the interrupt is there, how the CPU does know which particular device initiated this interrupt. So the first question is evaluation of the peripheral device or looking at which peripheral device has generated the interrupt. Now the second important question is that usually there would be a number of interrupts simultaneously available. So if there are a number of interrupts then there should be a mechanism by which we could just resolve that which particular interrupt should be serviced first. So there should be some priority mechanism.

Design Issues

There are two design issues:

1. Device Identification
2. Priority mechanism

Device Identification

In this issue different mechanisms could be used.

- Multiple interrupt lines
- Software Poll
- Daisy Chain

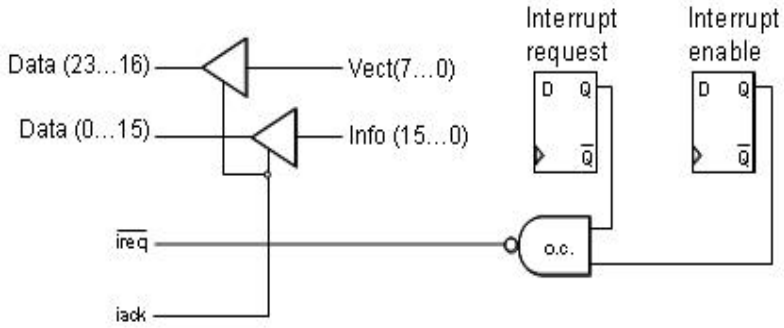
1. Multiple Interrupt Line

This is the most straight forward approach, and in this method, a number of interrupt lines are provided between the CPU and the I/O module. However, it is impractical to dedicate more than a few bus lines or CPU pins to interrupt lines. Consequently, even if multiple lines are used, it is

likely that each line will have multiple I/O modules attached to it. Thus on each line, one of the other technique would still be required.

2. Software Poll

CPU polls to identify the interrupting module and branches to an interrupt service routine on detecting an interrupt. This identification is done using special commands or reading the device status register. Special command may be a test I/O. In this case, CPU raises test I/O and places the address of a particular I/O module on the address line. If I/O module sets the interrupt then it responds positively. In the case of an addressable status register, the CPU reads the status register of each I/O module to identify the interrupting module. Once the correct module is identified, the CPU branches to a device service routine which is specific to that particular device.



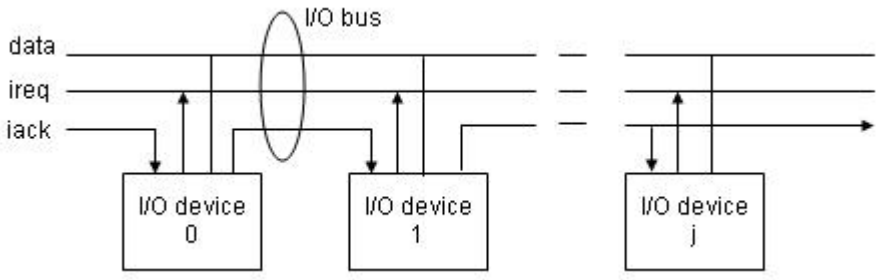
Simplified Interrupt Circuit for an I/O Interface

For above two techniques the implementation might require some hardware. The hardware would be specific to the processor which is being used. For example, for the case of SRC, simple hardware machanism is indicated. Now the basic technique is handshaking and in this case of handshaking, the peripheral device would initiate an interrupt. This interrupt needs to be enabled. We will have a mechanism of ANDing the two signals. One is interrupt enable and other is interrupt request. Now these two requests would be passed on the CPU. The CPU passes on the acknowledge signal to the device. The acknowledge signal is shared and it goes on to different devices.

The information about interrupt vector is given in 8-bits, from bit 0 to 7, which is translated to bit 16 to 23 on the data bus. Now the other 16-bits, from 0 to 15 are mapped to the data lines from 0 to 15. Now both of these are available through the tri-state buffers, which would be enabled through interrupt acknowledge.

3. Daisy Chain

The wired or interrupt signal allows several devices to request interrupt simultaneously. However, for proper operation one and only one requesting device must receive an acknowledge signal, otherwise if we have more than one devices, we would have a data bus contention and the interrupt information would not be resolved. The usual solution is called a daisy chain. Assuming that if we have jth devices requesting for interrupt then first device 0 would receive the acknowledge signal, so therefore, iack0=iack. The next device would only receive an acknowledge i.e., the jth device would receive an acknowledge if the previous device that means j-1 does not have an enabled interrupt request, that means interrupt was not initiated by the



previous device. Now the figure shows this concept in the form of a connection from device 0 to 1. From 0, we see the acknowledge is generated for device 1, device 1 generates acknowledge for device2 and so on. So this signal propagates from one device to other device. Logically we could write it in the form of equation:

$$iack_j = iack_{j-1} \wedge (req_{j-1} \wedge enb_{j-1})$$

As we said that the previous device should not have generated an interrupt, that means its interrupt was not enabled and therefore, it passes on the acknowledge signal from its output to he next device.

Disadvantages of Software Poll and Daisy Chain

The software poll has a disadvantage is that it consumes a lot of time, while the daisy chain is more efficient. The daisy chain has the disadvantage that the device nearest to the CPU would have highest priority. So, usually those devices which require higher priority would be connected nearer to the CPU. Now in order to get a fair chance for other devices, other mechanisms could be initiated or we could say that we could start instead of device 0 from that device where the CPU finishes the last interrupt and could have a cyclic provision to different devices.

Interrupt Handler Software

Example using SRC

(Read from Book, Jordan page395)

CICTL : The Control Register

```
;Getline is called with return address in R31 and a pointer to a
;character buffer in R1. It will input characters up to a carriage
;return under interrupt control, setting Done to -1 when complete.
CR      .equ    13          ;ASCII code for carriage return.
CIvec   .equ    01F0H       ;Character input interrupt vector address.
Bufp:   .dw     1           ;Pointer to next character location.
Save:   .dw     2           ;Save area for registers on interrupt.
Done:   .dw     1           ;Flag location is -1 if input complete.
GetIn:  st      r1, Bufp    ;Record pointer to next character.
    edi                     ;Disable interrupts while changing mask.
    la      r2, 1F1H        ;Get vector address and device enable bit
    st      r2, CICTL       ;and put into control register of device.
    la      r3, 0           ;Clear the
    st      r3, Done        ;line input done flag.
    een                     ;Enable Interrupts
    br      r31             ;and return to caller.
   .org     CIvec           ;Start handler at vector address.
    str     r0, Save        ;Save the registers that
    str     r1, Save+4      ; will be used by the interrupt handler.
    ldr     r1, Bufp        ;Get pointer to next character position.
    ld      r0, CIN         ;Get the character and enable next input.
    st      r0, 0(r1)       ;Store character in line buffer.
    addi    r1, r1, 4       ;Advance pointer and
    str     r1, Bufp        ; store for next interrupt.
    lar     r1, Exit        ;Set branch target.
    addi r0,r0, -CR         ;Carriage return? addi with minus CR.
    brnz r1, r0             ;Exit if not CR, else complete line.
    la      r0, 0           ;Turn off input device by
    st      r0, CICTL       ; disabling its interrupts.
    la      r0, -1          ;Get a -1 indicator, and
    str     r0, Done        ; report line input complete.
Exit:   ldr r0, Save        ;Restore registers
    ldr     r1, Save+4      ; of interrupted program.
    rfi                     ;Return to interrupted program.
```

Example using FALCON-A

As an example of interrupt-driven I/O, consider an output device, such as a parallel printer connected to the FALCON-A CPU. Now suppose that we want to print a document while using an application program like a word processor or a spread sheet. In this section, we will explain the important aspects of hardware and software for implementing an interrupt driven parallel printer interface for the FALCON-A. During this discussion, we will also explain the differences and similarities between this interface and the one discussed earlier. To make things simple, we have made the assumption that only one interrupt pin is available on the FALCON-A, and only one interrupt is possible at a given time with this CPU. Implications of allowing only one interrupt at a time are that

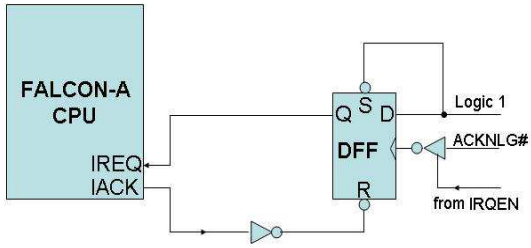
- No NMI is possible
- No nesting of interrupts is possible
- No priority structure needed for multiple devices
- No arbitration needed for simultaneous interrupts
- No need for vectored interrupts, therefore, no need of interrupt vectors and interrupt vector tables
- Effect of software initiated interrupts and internal interrupts (exceptions) has to be ignored in this discussion

Along with the previous assumption, the following assumptions have also been used:

- Hardware sets and clears the interrupt flag, in addition to handling other things like saving PC, etc.
- The address of the ISR is stored at absolute address 2 in memory.
- The ISR will set up a stack in the memory for saving the CPU's environment
- One ASCII character stored per 16-bit word in the FALCON-A's memory and one character transferred during a 16-bit transfer.
- The calling program will call the ISR for printing the first character through the printer driver.
- Printer will activate ACKNLG# only when not BUSY.

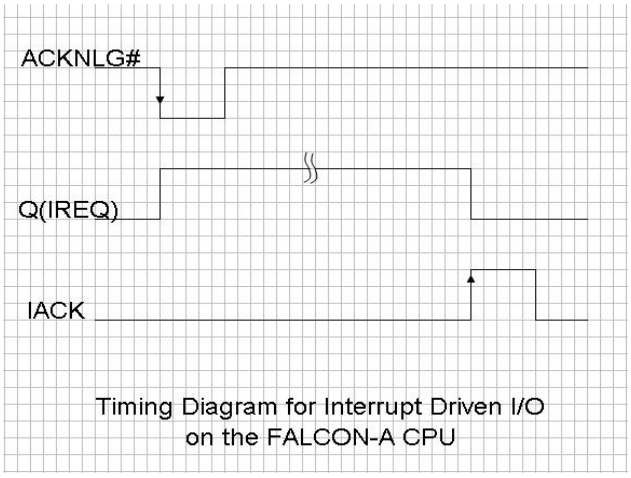
Interrupt Hardware:

The logic diagram for the interrupt hardware is shown in the Figure. The interrupt request is synchronized by handshaking signals, called IREQ and IACK. The timing diagram for the handshaking signals used in the interrupt driven I/O is shown in the next Figure. The printer will assert IREQ as soon as the ACKNLG# signal goes low (i.e. as soon as the printer is ready to accept new data) provided that IREQN=1. The processor will complete the current instruction and respond by executing the interrupt service routine. The inverting tri-state buffer at the clock input of the D flip flop is enabled by IRQEN. This will make sure that after the current print job is complete, additional requests on IREQ are disabled. This can happen as a result of the printer being available even through the user may not have requested a print operation. The IACK line from the CPU is connected to the asynchronous reset, R, of the D flip flop so that the same interrupt request from the printer is not presented again to the CPU. The asynchronous set input of the D flip flop, labeled S in the diagram, is permanently connected to logic 1.



Logic Diagram for Interrupt Driven I/O on the FALCON-A CPU

This will make sure that the flip flop will never be set asynchronously. The D input is also permanently connected to logic 1, as a result of which the flip flop will always be set synchronously in response to ACKNLG# provided IRQEN=1. Recall that IRQEN is bit 4 on the centronics control port at logical address 2, and this is mapped onto address 60 of the FALCON-A's I/O space. The rest of the hardware is case of the same as in the case of the programmed I/O example.

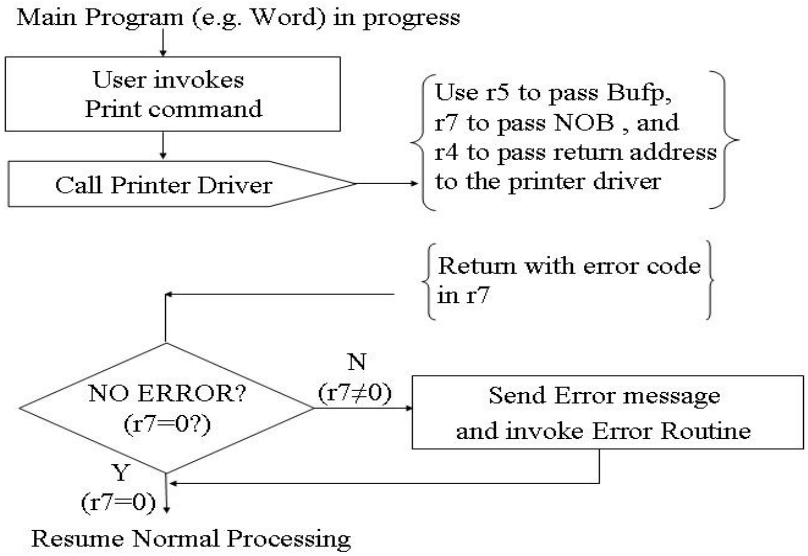


Timing Diagram for Interrupt Driven I/O on the FALCON-A CPU

Interrupt Software:

Our software for the interrupt driven printer example consists of three parts:
1). Dummy calling program
2). Printer Driver
3). ISR

We are assuming that normal processing is taking place[19] e.g., a word processor is executing. The user wants to print a document. This document is placed in a buffer by the word processor. This buffer is usually present somewhere else in the memory. The responsibility of the calling



program is to pass the number of bytes to be printed and the starting address of the buffer where these bytes are stored to the printer driver. The calling program can also be called the main program.

[19] Since only one interrupt is possible, a question may arise about the way the print command is presented to the word processor. It can be assumed that polling is used for the input device in this case.

```
; filename: Example_Falcon-A .asmfa
;This program sends a single character
;to a FALCON-A parallel printer
;using an interrupt driven I/O interface
;
; Notes:
; 1. 8-bit printer data bus connected to
;    D<7..0> of the FALCON-A (remember big-endian)
;    Thus, the printer actually uses addresses 57, 59 & 61
;
; 2. one character per 16-bits of data xfered ;
;
    .org 0
    jump [main]
a4ISR:    .sw  beginISR
a4PD:     .sw  Pdriver
dv1:            .sw 1024
dv2:            .sw 40
Bufp:     .dw  1
NOB:      .dw  1
PB:       .dw 1
temp:     .dw 6
;
; Dummy Calling Program, e.g., a word processor
;
    .org 32
main:     load r6, [a4PD]        ;r6 holds address of printer driver
;
; user invokes print command here
;
    load r5, [dv1]          ;Prepare registers for passing
    load r7, [dv2]          ; information about print buffer.
;
;
; call printer driver
;
    call r4, r6
; Handle error conditions, if any , upon return.
; Normal processing resumes
;
    halt
```

Suppose that the total number of bytes to be printed are 40. (They are placed in a buffer having the starting address 1024.) When the user invokes the print command, the calling program calls the printer driver and passes these two parameters in r7 and r5 respectively. The return address of the calling program is stored in r4. A dummy calling program code is given below. Bufp, NOB, PB, and temp are the spaces reserved in memory for later use in the program. The first instruction is **jump [main].** It is stored at absolute memory address 0 by using the **.org 0** directive. It will transfer control to the main program. The first instruction of the main program is

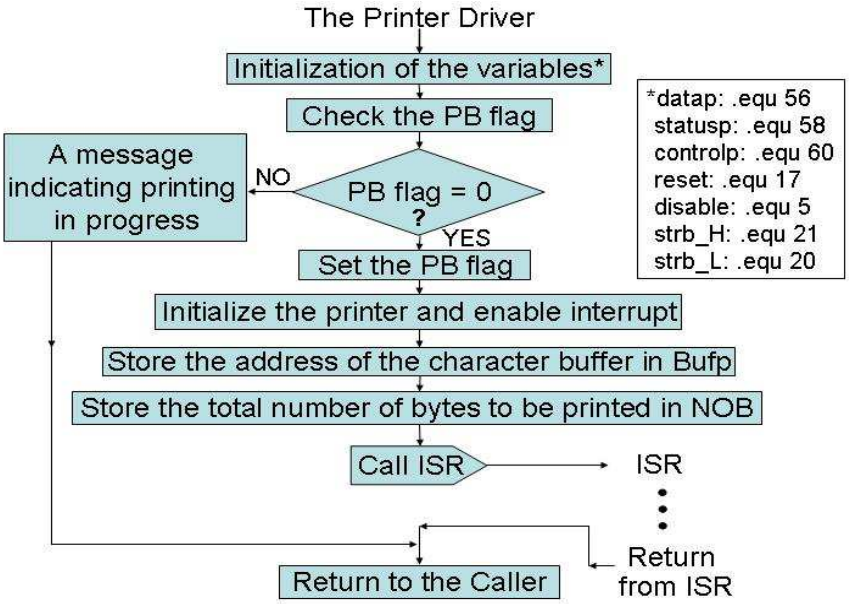
placed at address "**main**", which is the entry point in this example. Note that the entry point is different in this case from the reset address, which is address 0 for the FALCON-A. Also note that the address of the first instruction in the printer driver is stored at address "**a4PD"** using the **.sw** directive. This value is then brought into r6. The main program calls the printer driver by using the instruction **call r4, r6.** In an actual program, after returning from the printer driver, the normal processing resumes and if there are any error conditions, they will be handled at this point. Next, consider the code for the printer driver, shown in the attached text box.

The printer driver is loaded at address 50. Initialization of the variables includes setting of port addresses, variables for the STROBE# pulse, initializing the printer and enabling its IRQEN. The variables can be defined anywhere in the program because they reserve no memory space. When the printer driver starts, the PB flag is tested to make sure that a previous print job is not in progress. If so, the ISR is not invoked and a message is returned to the main program indicating that printing is in progress. This may display a "printer busy" icon on the user's screen, or cause some other appropriate action. If the printer is available, it is initialized by the driver.

The following activities are also performed by the driver (see the attached flow chart also).

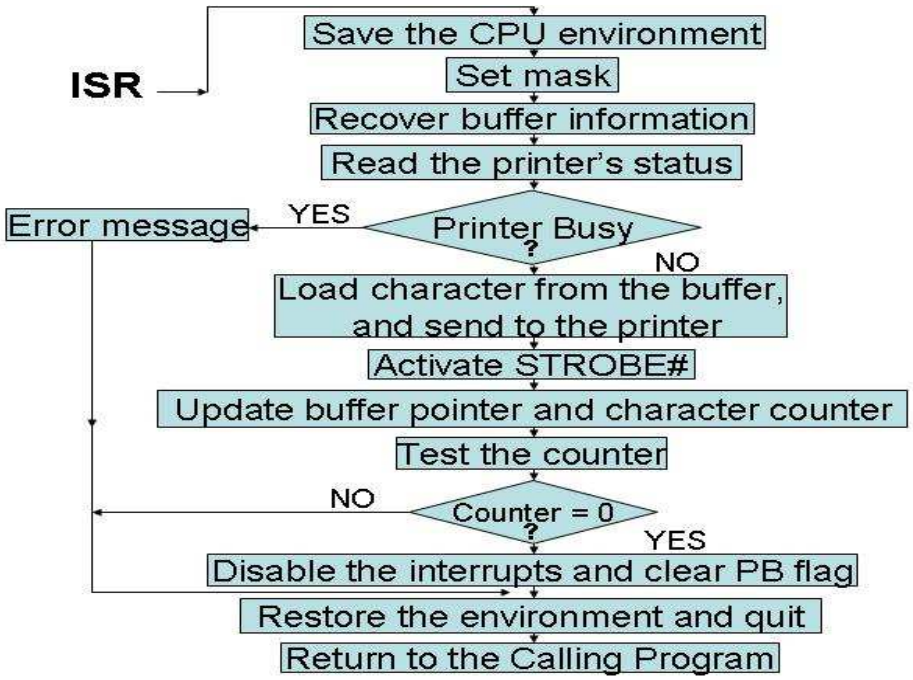
- Set port addresses
- Set up variables for the STROBE# puls
- Initialize printer and enable its IRQEN.
- Set up printer ISR by pointing to the buffer and initializing counter
- Make sure that the previous print job is not in progress
- Set PB flag to block further print jobs till current one is complete
- Invoke ISR for the first time
- Pass error message to main program if ISR reports an error
- Return to main program

The code and flow chart for the interrupt service routine (ISR) are discussed in the next few paragraphs.



```
; Printer driver
;
        .org 50                 ; starting address of Printer driver
;
datap:          .equ 56
statusp:        .equ 58
controlp:       .equ 60
;
reset:          .equ 17         ; or 11h
; used to set unidirectional, enable interrupts,
; auto line feed, and strobe high
disable:        .equ 5
;
strb_H:         .equ 21         ; or 15h
strb_L:         .equ 20         ; or 14h
;
; check PB flag first, if set,
; return with message.
;
Pdriver: load r1, [PB]
        jnz r1, [message]
        movi r1, 1
        store r1, [PB]          ; a 1 in PB indicates Print In Progress
        movi r1, reset          ; use r1 for data xfer
        out r1, controlp
        store r5, [Bufp]
        store r7, [NOB]
;
;
        int
;
        jump [finish]
message: nop                    ; in actual situation, put a message routine here
                                ;to indicate print in progress
finish:  ret r4
;
```

We have assumed that the address of the ISR is stored at absolute memory address 2 by the operating system. One way to do that is by using the **.sw** directive (as done in the dummy calling program). The symbol **sw** stands for "storage of word". It enables the user to identify storage for a constant, or the value of a variable, an address or a label at a fixed memory location during the assembly process.



These values become part of the binary file and are then loaded into the memory when the binary file is loaded and executed. In response to a hardware interrupt or the software interrupt **int,** the control unit of the FALCON-A CPU will pick up the address of the first instruction in the ISR from memory location 2, and transfer control to it. This effectively means that the behavioral RTL of the **int** instruction will be as shown below:

int IPC← PC, PC ← M[2], IF ← 0

The IPC register in the CPU is a holding place for the current value of the PC. It is invisible to the programmer. Since the **iret** instruction should always be the last instruction in every ISR, its behavior RTL will be as shown below:

iret PC ← IPC, IF ← 1

The saving and restoring of the other elements of the CPU environment like the general purpose registers should be done within the ISR. The five **store** instructions at the beginning are used to save these registers into the memory block starting at address **temp**, and the five **load** instructions at the end are used to restore these registers to their original values.

```
;  ISR starts here
    .org 100
beginISR: movi r6, temp
    store r1, [r6]
    store r3, [r6+2]
    store r4, [r6+4]
    store r5, [r6+6]
    store r7, [r6+8]
    movi r3, 1
    shiftl r3,r3,7          ; to set mask to 0080h
    load r5, [Bufp]         ; not necessary to use r5 & r7 here
    load r7, [NOB]          ; using r7 as character counter
    in r1, statusp
    and r1,r1,r3            ; test if BUSY = 1 ?
    jnz r1, [error]         ; error if BUSY = 1
    load r1, [r5]           ; get char from printer buffer
    out r1, datap
    movi r1, strb_L
    out r1, controlp
    movi r1, strb_H
    out r1, controlp
    addi r5, r5, 2
    store r5, [Bufp]        ; update buffer pointer
    subi r7, r7, 1          ; update character counter
    store r7, [NOB]
    jz r7, [suspend]
    jump [last]
suspend: store r7, [PB]     ; clear PB flag
    movi r1, disable        ; disable future interrupts till
    out r1, controlp        ; printer driver called again
    jump [last]
error: movi r7, -1          ; error code in r7
; other error codes go here
;
last: load r1, [r6]
    load r3, [r6+2]
    load r4, [r6+4]
    load r5, [r6+6]
    load r7, [r6+8]
    iret
    .end
```

After setting the mask to 80h in r3, the current value of the buffer pointer and the number of bytes to be printed are brought from the memory into r5 and r7 respectively. After a byte is printed, these values are updated in the memory for use by the ISR when it is invoked again. The rest of the code in the ISR is the same as it was in case of the programmed I/O example. Note that we are testing the printer's BUSY flag within the ISR also. However, the difference here is

that this testing is being done for a different reason, and it is done only once for each call to the ISR.

Memory Map for our ISR

0	Entry Point
2	Address of ISR
4	Data and Pointer Area
32	Main Program (Dummy Calling Program)
50	Printer Driver
100	ISR
	. . .
1024	Print Buffer
	. . .

The memory map for this program is as shown in the Figure. The point to be noted here is that the ISR can be loaded anywhere in the memory but its address will be present at memory location 2 i.e. M[2].

Lecture No. 29

FALSIM

Reading Material

Handouts Slides

Summary

- Introduction to FALSIM
- Preparing source files for FALSIM
- Using FALSIM
- FALCON-A assembly language techniques

Introduction to FALSIM:

FALSIM is the name of the software application which consists of the FALCON-A assembler and the FALCON-A simulator. It runs under Windows XP.

FALCON-A Assembler:

Figure 1 shows a snapshot of the graphical user interface (GUI) for the FALCON-A Assembler. This tool loads a FALCON-A assembly file with a (.asmfa) extension and parses it. It shows the parsed results in an error log, lets the user view the assembled file's contents in the file listing and also provides the features of printing the machine code, an Instruction Table and a Symbol Table to a FALCON-A listing file. It also allows the user to run the FALCON-A Simulator.

The FALCON-A Assembler source code has two main modules, the 1st-pass module and the 2nd-pass module. The 1st-pass module takes an assembly file with a (.asmfa) extension and processes the file contents. It then generates a Symbol Table which corresponds to the storage of all program variables, labels and data values in a data structure at the implementation level. The Symbol Table is used by the 2nd-pass module. Failures of the 1st-pass are handled by the assembler using its exception handling mechanism.

The 2nd-pass module sequentially processes the .asmfa file to interpret the instruction op-codes, register op-codes and constants using the Symbol Table. It then produces a list file with a .lstfa extension independent of successful or failed pass. If the pass is successful a binary file with a .binfa extension is produced which contains the machine code for the program contained in the assembly file.

FALCON-A Simulator:

Figure 6 shows a snapshot of the GUI for the FALCON-A Simulator. This tool loads a FALCON-A binary file with a (.binfa) extension and presents its contents into different areas of the simulator. It allows the user to execute the program to a specific point within a time frame or just executes it, line by line. It also allows the user to view the registers, I/O port values and memory contents as the instructions execute.

FALSIM Features:

The FALCON-A Assembler provides its user with the following features:

Select Assembly File: Labeled as "**1**" in Figure 1, this feature enables the user to choose a FALCON-A assembly file and open it for processing by the assembler. *Assembler Options:* Labeled as "**2**" in Figure 1.

- *Print Symbol Table*

This feature, if selected, writes the Symbol Table (produced after the execution of the 1st-pass of the assembler) to a FALCON-A list file with an extension of (.lstfa). The Symbol Table includes variables, addresses and labels with their respective values.

- *Print Instruction Table*

This feature, if selected, writes the FALCON-A instructions along with their op-codes at the end of the list file.

List File: Labeled as "**3**", in Figure 1, the List File feature gives a detailed insight of the FALCON-A listing file, which is produced as a result of the execution of the 1st and 2nd-pass. It shows the Program Counter value in hexadecimal and decimal formats along with the machine code generated for every line of assembly code. These values are printed when the 2nd-pass is completed.

Error Log: The Error Log is labeled as "**4**" in Figure 1. It informs the user about the errors and their respective details, which occurs in any of the two passes of the assembler. The size of this window can be changed by dragging the boundary line up or down.

Highlight: This feature is labeled as "**5**" in Figure 1 and helps the user to search for a certain input with the options of searching with "**match whole**" and "**match any**" parts of the string. The search also has the option of checking with/without considering "**case-sensitivity**". It searches the List File area and highlights the search results using the yellow color. It also indicates the total number of matches found.

Start Simulator: This feature is labeled as "**6**" in Figure 1. The FALCON-A Simulator is run using the FALCON-A Assembler's "Start Simulator" option. Its features are detailed as follows:

Load Binary File: The button labeled as "**11**" in Figure 6, allows the user to choose and open a FALCON-A binary file with a (.binfa) extension. When a file is being loaded into the simulator all the register, constants (if any) and memory values are set.

Registers: The area labeled as "**12**" in Figure 6. enables, the user to see values present in different registers before, during and after execution.

Instruction: This area is labeled as "**13**" in Figure 6 and contains the value of PC, address of an instruction, its representation in Assembly, the Register Transfer Language, the op-code and the instruction type.

I/O Ports: I/O ports are labeled as "**14**" in Figure 6. These ports are available for the user to enter input operation values and visualize output operation values whenever an I/O operation takes place in the program. The input value for an input operation is given by the user before an instruction executes. The output values are visible in the I/O port area once the instruction has successfully executed.

Memory: The memory is divided into two areas and is labeled as "**15**" in Figure 6, to facilitate the view of data stored at different memory locations before, during and after program execution.

Processor's State: Labeled as "**16**" in Figure 6, this area shows the current values of the Instruction Register and the Program Counter while the program executes.

Highlight: The highlight option for the FALCON-A simulator is labeled as "**17**" in Figure 6. This feature is similar to the way the highlight feature of the FALCON-A Assembler works. It offers to highlight the search string which is entered as an input, with the "All " and " Part " option. The results of the search are highlighted using the yellow color. It also indicates the total number of matches.

The following is a description of the options available on the button panel labeled as "**18**" in Figure 6.

Single Step: "Single Step" lets the user execute the program, one instruction at a time. The next instruction is not executed unless the user does a "single step" again. By default, the instruction to be executed will be the one next in the sequence. It changes if the user specifies a different PC value using the Change PC option (explained below).

Change PC: This option lets the user change the value of PC (Program Counter). By changing the PC the user can execute the instruction to which the specified PC points. The value in the PC must be an even address.

Execute: By choosing this button, the user is able to execute the loaded program with the options of execution with/without breakpoint insertion. In case of breakpoint insertion, the user has the option to choose from a list of valid breakpoint values. It also has the option to set a limit on the time for execution. This "Max Execution Time" option restricts the program execution to a time frame specified by the user.

Change Register: Using the Change Register feature, the user can change the value present in a particular register.

Change Memory Word: This feature enables the user to change values present at a particular memory location.

Display Memory: Display Memory shows an updated memory area, after a particular memory location other than the pre-existing ones is specified by the user.

Change I/O: Allows the user to give an I/O port value if the instruction to be executed requires an I/O operation. Giving in the input in any one of the I/O ports areas before instruction execution, indicates that a particular I/O operation will be a part of the program and it will have an input from some source. The value given by the user indicates the input type and source.

Display I/O: Display I/O works in a manner similar to Display Memory. Here the user specifies the starting index of an I/O port. This features displays the I/O ports stating from the index specified.

2. Preparing Source Files for FALSIM:

In order to use the FALCON-A assembler and simulator, FALSIM, the source file containing assembly language statements and directives should be prepared according to the following guidelines:

- The source file should contain ASCII text only. Each line should be terminated by a carriage return. The extension **.asmfa** should be used with each file name. After assembly, a list file with the original filename and an extension **.lstfa,** and a binary file with an extension **.binfa** will be generated by FALSIM.
- Comments are indicated by a semicolon (;) and can be placed anywhere in the source file. The FALSIM assembler ignores any text after the semicolon.
- Names in the source file can be of one of the following types:
- Variables: These are defined using the **.equ** directive. A value must also be assigned to variables when they are defined.
- Addresses in the "data and pointer area" within the memory: These can be defined using the **.dw** or the **.sw** directive. The difference between these two directives is that when **.dw** is used, it is not possible to store any value in the memory. The integer after **.dw** identifies the number of memory words to be reserved starting at the current address. (The directive **.db** can be used to reserve bytes in memory.) Using the **.sw** directive, it is possible to store a constant or the value of a name in the memory. It is also possible to use pointers with this directive to specify addresses larger than 127. Data tables and jump tables can also be set up in the memory using this directive.
- Labels: An assembly language statement can have a unique label associated with it. Two assembly language statements cannot have the same name. Every label should have a colon (:) after it.
- Use the **.org 0** directive as the first line in the program. Although the use of this line is optional, its use will make sure that FALSIM will start simulation by picking up the first instruction stored at address 0 of the memory. (Address 0 is called the reset address of the processor). A **jump [first]** instruction can be placed at address 0, so that control is transferred to the first executable statement of the main program. Thus, the label **first**

serves as the identifier of the "entry point" in the source file. The **.org** directive can also be used anywhere in the source file to force code at a particular address in the memory.

- Address 2 in the memory is reserved for the pointer to the Interrupt Service Routine (ISR). The **.sw** directive can be used to store the address of the first instruction in the ISR at this location.
- Address 4 to 125 can be used for addresses of data and pointers[20]. However, the main program must start at address 126 or less[21], otherwise FALSIM will generate an error at the **jump [first]** instruction.
- The main program should be followed by any subprograms or procedures. Each procedure should be terminated with a **ret** instruction. The ISR, if any, should be placed after the procedures and should be terminated with the **iret** instruction.
- The last line in the source file should be the **.end** directive.
- The **.equ** directive can be used anywhere in the source file to assign values to variables.
- It is the responsibility of the programmer to make sure that code does not overwrite data when the assembly process is performed, or vice versa. As an example, this can happen if care is not exercised during the use of the **.org** directive in the source file.

3. Using FALSIM:

- To start FALSIM (the FALCON-A assembler and simulator), double click on the FALSIM icon. This will display the assembler window, as shown in the Figure 1.

- Select one or both assembler options shown on the top right corner of the assembler window labeled as "2". If no option is selected, the symbol table and the instruction table will not be generated in the list (.lstfa) file.

- Click on the select assembly file button labeled as "1". This will open the dialog box as shown in the Figure 2.

- Select the path and file containing the source program that is to be assembled.

- Click on the open button. FALSIM will assemble the program and generate two files with the same filename, but with different extensions. A list file will be generated with an extension .lstfa, and a binary (executable) file will be generated with an extension .binfa. FALSIM will also display the list file and any error messages in two separate panes, as shown in Figure 3.

- Double clicking on any error message highlights and displays the corresponding erroneous line in the program listing window pane for the user. This is shown in Figure 4. The highlight feature can also be used to display any text string, including statements with errors in them. If the assembler reported any errors in the source file, then these errors should be corrected and the program should be assembled again before simulation can be done. Additionally, if the source file had been assembled correctly at an earlier occasion, and a correct binary (.binfa) file exists, the simulator can be started directly without performing the assembly process.

-

---

[20] Any address between 4 and 14 can be used in place of the displacement field in load or store instructions. Recall that the displacement field is just 5 bits in the instruction word.

[21] This restriction is because of the fact that the immediate operand in the **movi** instruction must fit an 8-bit field in the instruction word.

- To start the simulator, click on the start simulation button labeled as "6". This will open the dialog box shown in Figure 6.

- Select the binary file to be simulated, and click Open as shown in Figure 7. (It is also possible to open the file by double clicking on the file name in the "Open" window).

- This will open the simulation window with the executable program loaded in it as shown in Figure 8. The details of the different panes in this window were given in section 1 earlier. Notice that the first instruction at address 0 is ready for execution. All registers are initialized to 0. The memory contains the address of the ISR (i.e., 64h which is 100 decimal) at location 2 and the address of the printer driver at location 4. These two addresses are determined at assembly time in our case. In a real situation, these addresses will be determined at execution time by the operating system, and thus the ISR and the printer driver will be located in the memory by the operating system (called re-locatable code). Subsequent memory locations contain constants defined in the program.

- Click single step button labeled as "**19**". FALSIM will execute the **jump [main]** instruction at address 0 and the PC will change to 20h (32 decimal), which is the address of the first instruction in the main program (i.e., the value of main).

- Although in a real situation, there will be many instructions in the main program, those instructions are not present in the dummy calling program. The first useful instruction is shown next. It loads the address of the printer driver in r6 from the pointer area in the memory. The registers r5 and r7 are also set up for passing the starting address of the print buffer and the number of bytes to be printed. In our dummy program, we bring these values in to these registers from the data area in the memory, and then pass these values to the printer driver using these two registers. Clicking on the single step button twice, executes these two instructions.

- The execution of the call instruction simulates the event of a print request by the user. This transfers control to the printer driver. Thus, when the **call r4, r6** instruction is single stepped, the PC changes to 32h (50 decimal) for executing the first instruction in the printer driver.

- Double click on memory location 000A, which is being used for holding the PB (printer busy) flag. Enter a 1 and click the change memory button. This will store a 0001 in this location, indicating that a previous print job is in progress. Now click single step and note that this value is brought from memory location 000E into register r1. Clicking single step again will cause the **jnz r1, [message]** instruction to execute, and control will transfer to the message routine at address 0046h. The **nop** instruction is used here as a place holder.

- Click again on the single step button. Note that when the **ret r4** instruction executes, the value in r4 (i.e., 28h) is brought into the PC. The blue highlight bar is placed on the next instruction after the **call r4, r6** instruction in the main program. In case of the dummy calling program, this is the **halt** instruction.

- Double click on the value of the PC labeled as "**20**". This will open a dialog box shown below. Enter a value of the PC (i.e. 26h) corresponding to the **call r4, r6** instruction, so that it can be executed again. A "list" of possible PC values can also be pulled down using, and 0026h can be selected from there as well.

- Click single step again to enter the printer driver again.

- Change memory location 000A to a 0, and then single step the first instruction in the printer driver. This will bring a 0 in r1, so that when the next **jnz r1, [message]**

instruction is executed, the branch will not be taken and control will transfer to the next instruction after this instruction. This is **movi r1, 1** at address 0036h.

- Continue single stepping.

- Notice that a 1 has been stored in memory location 000A, and r1 contains 11h, which is then transferred to the output port at address 3Ch (60 decimal) when the **out r1, controlp** instruction executes. This can be verified by double clicking on the top left corner of the I/O port pane, and changing the address to 3Ch. Another way to display the value of an I/O port is to scroll the I/O window pane to the desired position.

- Continue single stepping till the **int** instruction and note the changes in different panes of the simulation window at each ste

- When the **int** instruction executes, the PC changes to 64h, which is the address of the first instruction in the ISR. Clicking single step executes this instruction, and loads the address of **temp** (i.e., 0010h) which is a temporary memory area for storing the environment. The five **store** instructions in the ISR save the CPU environment (working registers) before the ISR change them.

- Single step through the ISR while noting the effects on various registers, memory locations, and I/O ports till the **iret** instruction executes. This will pass control back to the printer driver by changing the PC to the address of the **jump [finish]** instruction, which is the next instruction after the **int** instruction.

- Double click on the value of the PC. Change it to point to the **int** instruction and click single step to execute it again. Continue to single step till the **in r1, statusp** instruction is ready for execution.

- Change the I/O port at address 3Ah (which represents the status port at address 58) to 80 and then single step the **in r1, statusp** instruction. The value in r1 should be 0080.

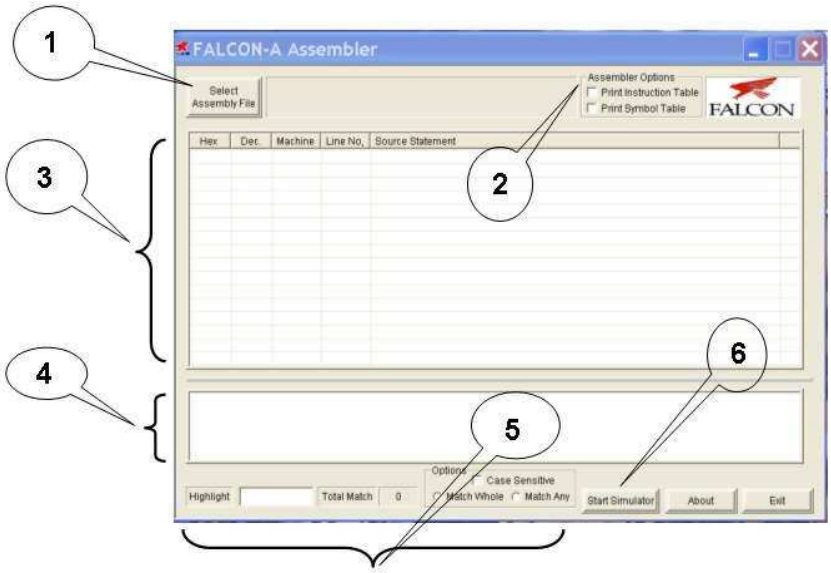


Figure 1

- Single step twice and notice that control is transferred to the **movi r7, FFFF**[22] instruction, which stores an error code of –1 in r1.

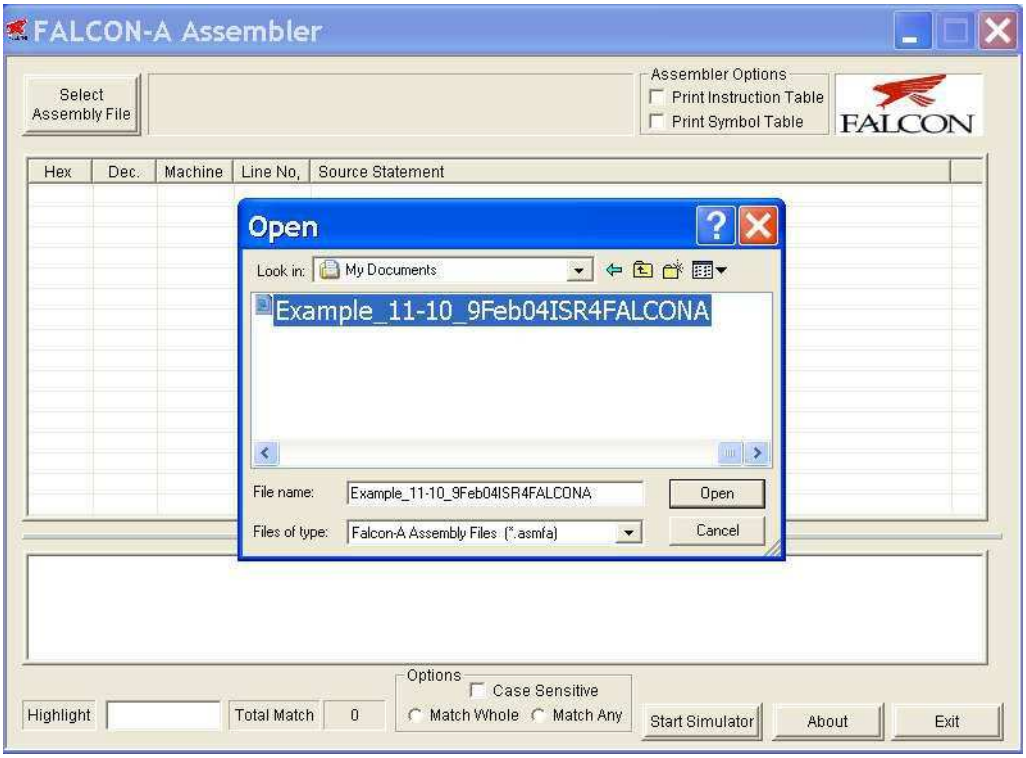


Figure 2

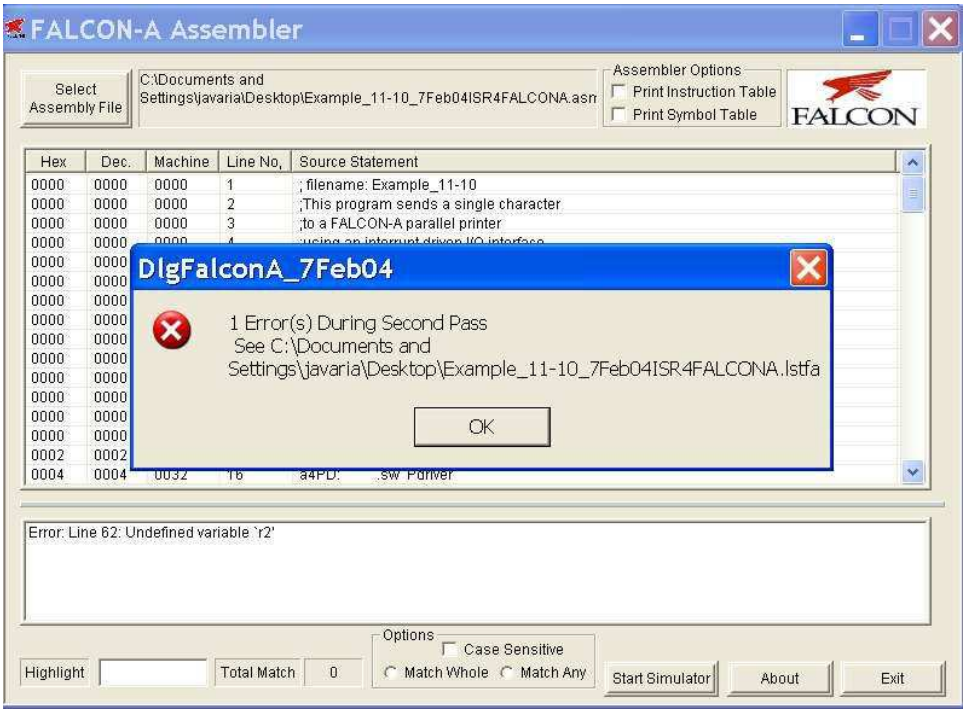


Figure 3

22 The instruction was originally **movi r7, -1.** Since it was converted to machine language by the assembler, and then reverse assembled by the simulator, it became **movi r7, FFFF.** This is because the machine code stores the number in 16-bits after sign-extension. The result will be the same in both cases.

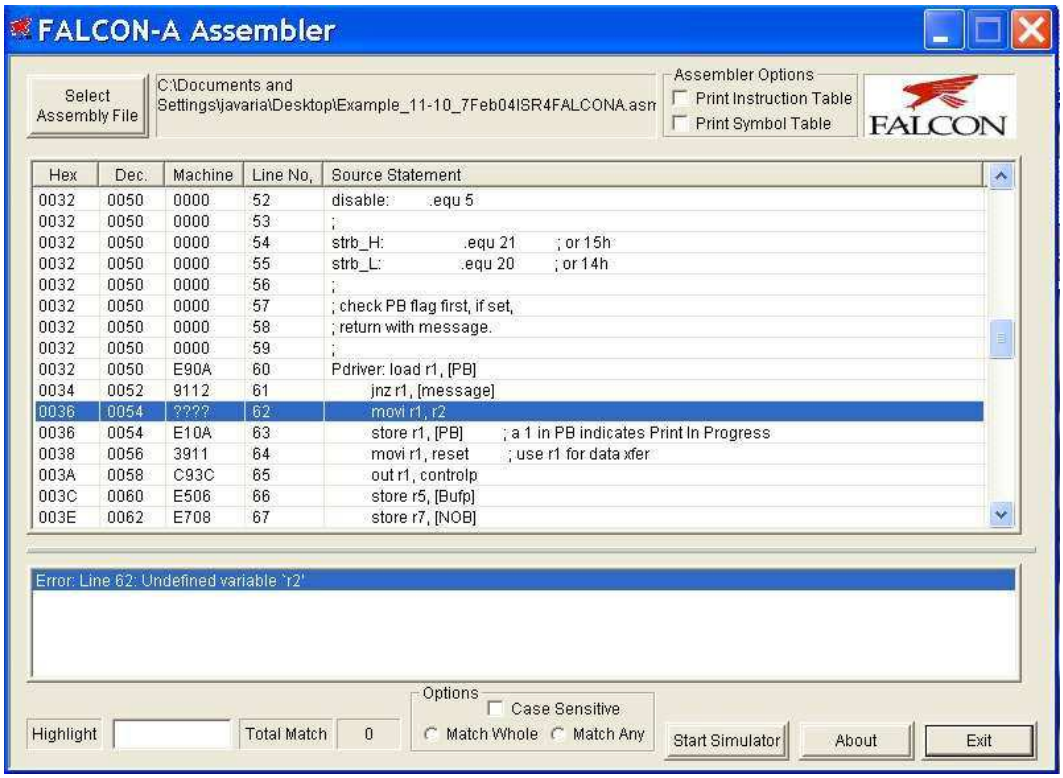


Figure 4

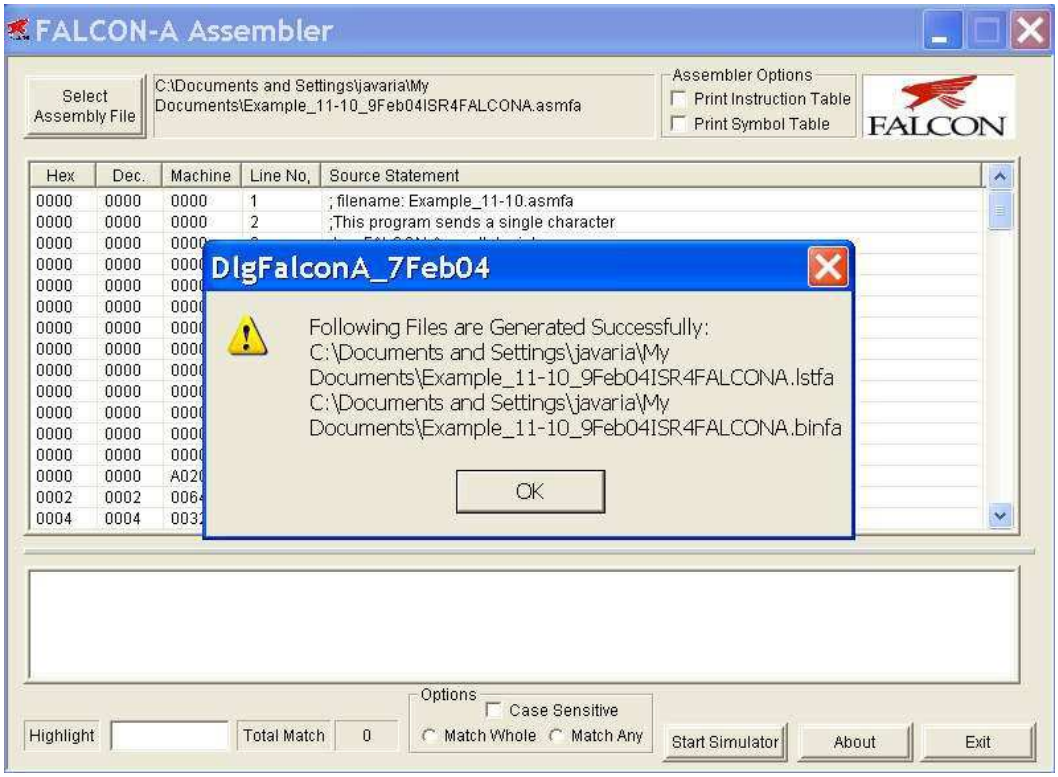


Figure 5

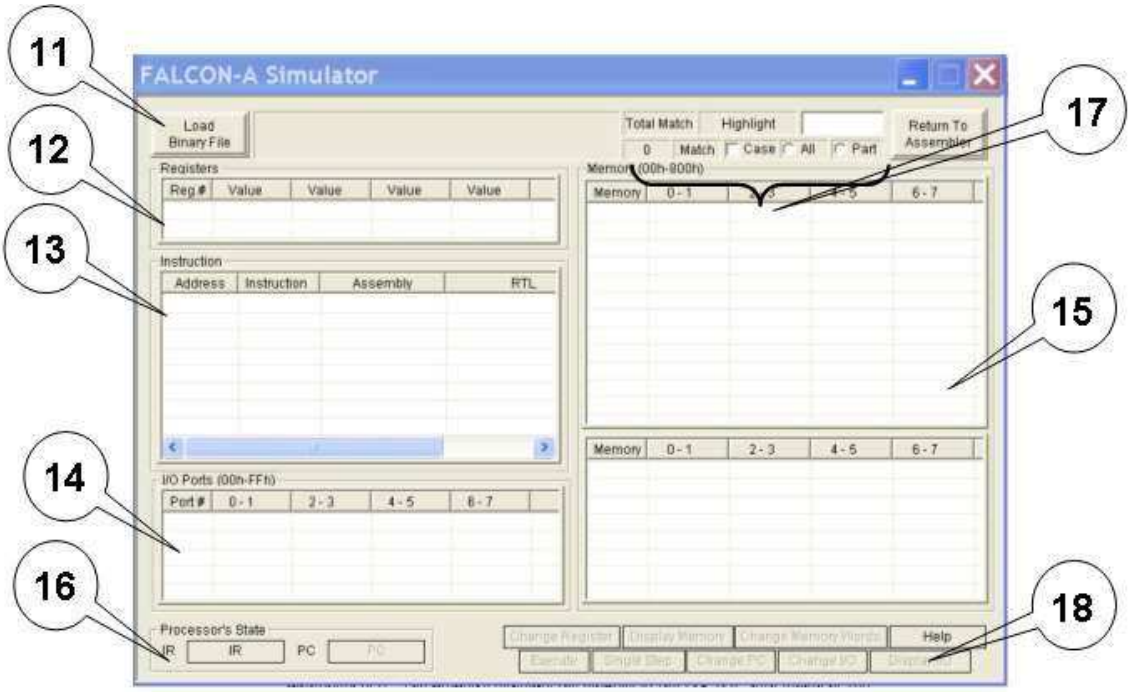


Figure 6

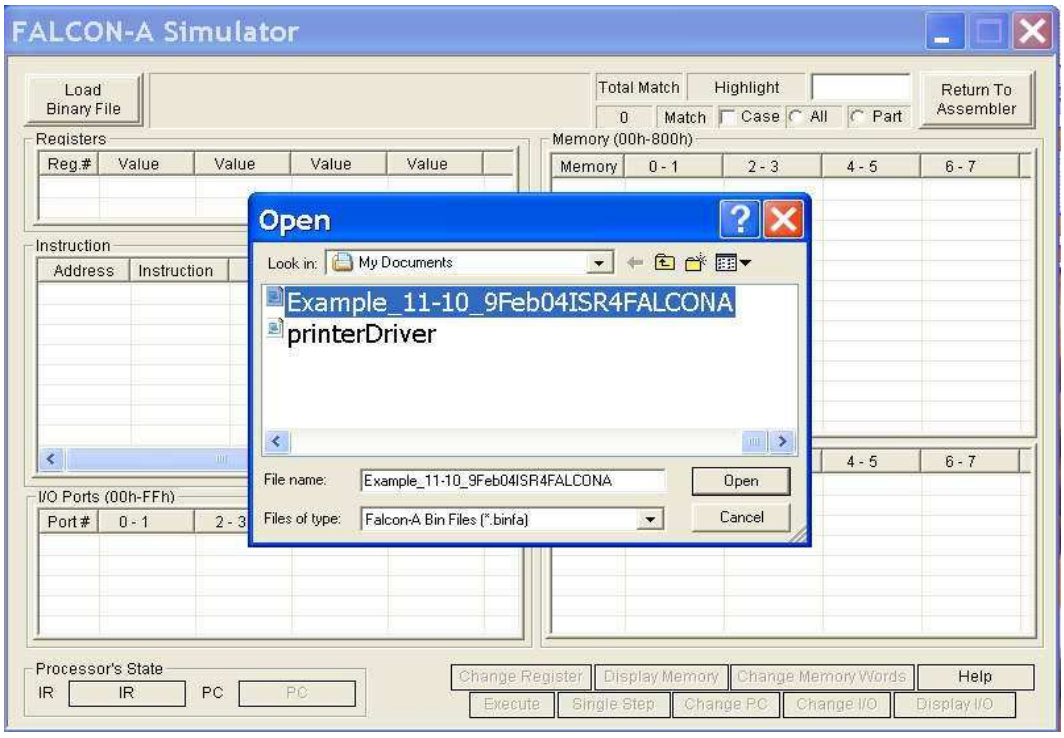


Figure 7

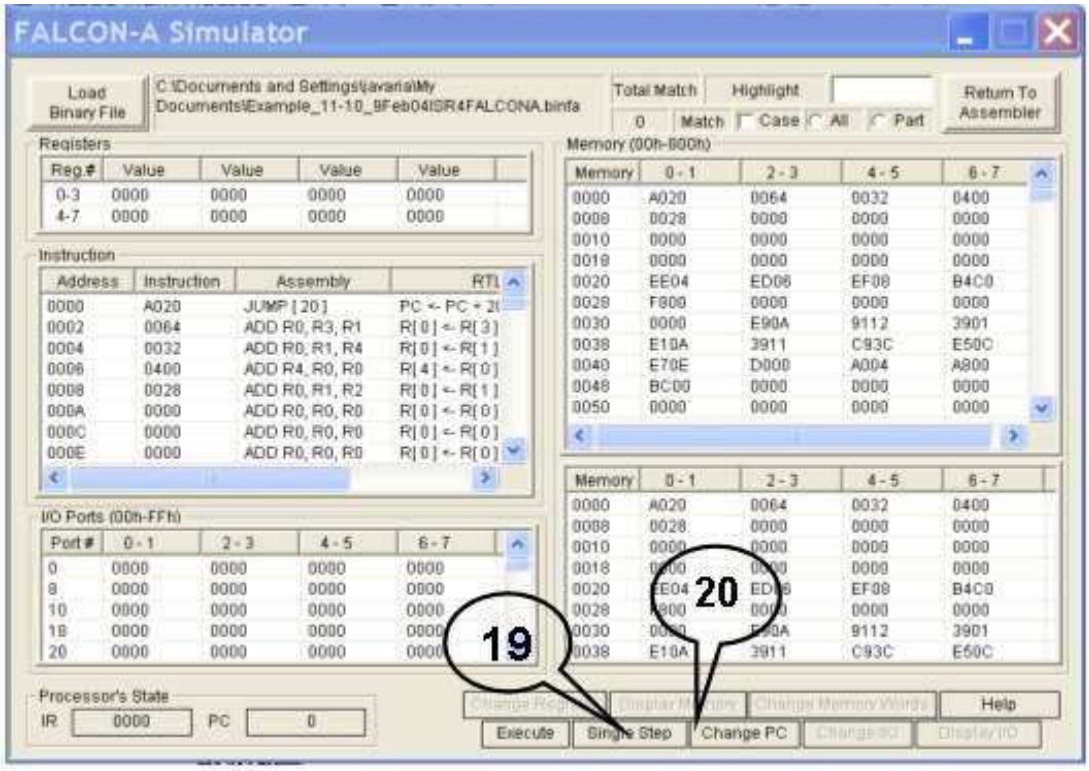


Figure 8

4. FALCON-A assembly language programming techniques:

- If a signed value, x, cannot fit in 5 bits (i.e., it is outside the range -16 to +15), FALSIM will report an error with a load r1, [x] or a store r1, [x] instruction. To overcome this problem, use movi r2, x followed by load r1, [r2].

- If a signed value, x, cannot fit in 8 bits (i.e., it is outside the range -128 to +127), even the previous scheme will not work. FALSIM will report an error with the movi r2, x instruction. The following instruction sequence should be used to overcome this limitation of the FALCON-A. First store the 16-bit address in the memory using the .sw directive. Then use two load instructions as shown below:

```
a:      .sw x load r2, [a]
        load r1, [r2]
```

  This is essentially a "memory-register-indirect" addressing. It has been made possible by the .sw directive. The value of a should be less than 15.

- A similar technique can be used with immediate ALU instructions for large values of the immediate data, and with the transfer of control (call and jump) instructions for large values of the target address.

- Large values (16-bit values) can also be stored in registers using the mul instruction combined with the addi instruction. The following instructions bring a 201 in register r1.

```
movi r2, 10
movi r3, 20
mul r1, r2, r3        ; r1 contains 200 after this instruction
addi r1, r1, 1        ; r1 now contains 201
```

- Moving from one register to another can be done by using the instruction **addi r2, r1, 0.**
- Bit setting and clearing can be done using the logical (and, or, not, etc) instructions.
- Using shift instructions (shiftl, asr, etc.) is faster that **mul** and **div,** if the multiplier or divisor is a power of 2.

Lecture No. 30
Interrupt Priority and Nested Interrupts

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 8
Computer Systems Design and Architecture — 8.3.3, 8.4

Summary

- Nested Interrupts
- Interrupt Mask
- DMA

Nested Interrupts

(Read from Book, Jordan Page 391)

Interrupt Mask

(Read from Book, Jordan Page 391)

Priority Mask

(Read from Book, Jordan Page 392)

Examples

Example # 1[23]

Assume that three I/O devices are connected to a 32-bit, 10 MIPS CPU. The first device is a hard drive with a maximum transfer rate of 1MB/sec. It has a 32-bit bus. The second device is a floppy drive with a transfer rate of 25KB/sec over a 16-bit bus, and the third device is a keyboard that must be polled thirty times per second. Assuming that the polling operation requires 20 instructions for each I/O device, determine the percentage of CPU time required to poll each device.

Solution:

The hard drive can transfer 1MB/sec or 250 K 32-bit words every second. Thus, this hard drive should be polled using at least this rate.

Using $1K=2^{10,}$ the number of CPU instructions required would be

$$250 \times 2^{10} \times 20 = 5120000 \text{ instructions per second.}$$

Percentage of CPU time required for polling is

$$(5.12 \times 10^6)/ (10 \times 10^6) = 51.2\%$$

[23] Adopted from [H&P org]

The floppy disk can transfer 25K/2= 12.5 x 2^{10} half-words per second. It should be polled with at least this rate. The number of CPU instructions required will be 12.5 x 2^{10} x 20 = 256,000 instructions per second.

Therefore, the percentage of CPU time required for polling is

$(0.256 \times 10^6)/ (10 \times 10^6) = 2.56\%$.

For the keyboard, the number of instructions required for polling is

30 x 20 = 600 instructions per second.

Therefore, the percentage of CPU time spent in polling is

$600 / (10 \times 10^6) = 0.006\%$

It is clear from this example that while it is acceptable to use polling for a keyboard or a floppy drive, it is very risky to use polling for the hard drive. In general, for devices with a high data rate, the use of polling is not adequate.

Example # 2[2]

a. What should be the polling frequency for an I/O device if the average delay between the time when the device wants to make a request and the time when it is polled, is to be at most 10 ms?

b. If it takes 10,000 cycles to poll the I/O device, and the processor operates at 100MHz, what % of the CPU time is spent polling?

c. What if th[24]e system wants to provide an average delay of 1msec?

Solution:

a. Assuming that the I/O requests are distributed evenly in time, the average time that a device will have to wait for the processor to poll is half the time between polling attempts. Therefore, to provide an average delay of 10 ms, the processor will have to poll every 20 ms, or 50 times per second.

b. If each polling attempt takes 10,000 cycles, then the processor will spend 500,000 cycles polling each second. The % of CPU time spent in polling is then
$(0.5 \times 10^6)/(100 \times 10^6)=0.5\%$

c. To provide an average delay of 1ms, the polling frequency must be increased. The processor will have to poll every 2ms, or 500 times per second. This will consume 5,000,000 cycles for polling. The % of CPU time spent polling then becomes 5/100=5%.

Example # 3[25]

What percentage of time will a 20MIPS processor spend in the busy wait loop of an 80-character line printer when it takes 1 msec to print a character and a total of 565 instructions need to be executed to print an 80 character line. Assume that two instructions are executed in the polling loop.

24 Adopted from [Schaum]

25 Adopted from [H&J]

Solution:
Out of the total 565 instructions executed to print a line, 80x2=160 are required for polling. For a 20MIPS processor, the execution of the remaining 405 instructions takes 405/ ($20x10^6$) = 20.25∝sec. Since the printing of 80 characters takes 80ms, (80-0.02025) =79.97msec is spent in the polling loop before the next 80 characters can be printed. This is 79.97/80=99.96% of the total time.

Example # 4[26]
Consider a 20 MIPS processor with several input devices attached to it, each running at 1000 characters per second. Assume that it takes 17 instructions to handle an interrupt. If the hardware interrupt response takes 1∝sec, what is the maximum number of devices that can be handled simultaneously?

Solution:
A service for one character requires 17/ ($20x10^6$) +1∝sec=1.85∝sec. Since each device runs at 1000 characters per second, 1.85 ms of handling time is required by each device every second. Therefore the maximum number of devices that can be handled is 1/ ($1.85x10^{-3}$) = 540.

Example # 5[27]
Assume that a floppy drive having a transfer rate of 25KB per second is attached to a 32 bit, 10MIPS CPU using an interrupt driven interface. The drive has a 16-bit data bus. Assume that the interrupt overhead is 20 instructions. Calculate the fraction of CPU time required to service this drive when it is active.

Solution:
Since the floppy drive has a 16-bit data bus, it can transfer two bytes at one time. Thus its transfer rate is 25/2 = 12.5K half-words (16-bits each) per second. This corresponds to an overhead of 20 instructions or 12.5K x 20 = 12.5 x 2^{10} x 20 = 256000 instructions per second.

Example # 6[28]
A processor with a 500 MHz clock requires 1000 clock cycles to perform a context switch and start an ISR. Assume each interrupt takes 10,000 cycles to execute the ISR and the device makes 200 interrupt requests per second. Also, assume that the processor polls every 0.5msec during the time when there are no interrupts. Further assume that polling an I/O device requires 500 cycles. Compute the following:

a. How many cycles per second does the processor spend handling I/O from the device if only interrupts are used?
b. What fraction of the CPU time is used in interrupt handling for part (a)?
c. How many cycles per second are spent on I/O if polling is also used with interrupts?
d. How often should the processor poll so that polling incurs the same overhead as interrupts?

26 Adopted from [H&J]
27 Adopted from [H&P org]
28 Adopted from [Schaum]

Solution:

a. The device makes 200 interrupt requests per second, each of which takes 10,000 + 2x1000 (context switching to the ISR and back from it)
= 12,000 cycles.

Thus, a total of 200x12,000=2,400,000 cycles per second are spent handling I/O using interrupts.

b. The percentage of the processor time used in interrupt handling is 2,400,000/(500x10$^{6)}$ or 0.48%.

c. There are 200 interrupt requests per second, or one interrupt request every 5 ms. Every interrupt consumes a total of 12,000 cycles, as calculated in part (a). For a 500 MHz CPU, this is

$12000 / (500 \times 10^6) = 24$ microseconds.

For 200 interrupts per second, this is 4.8 msec. This leaves 1000 - 4.8 = 995.2 msec for polling.

Since the processor polls once every 0.5 msec during the time when there is no interrupt, this corresponds to

995 / 0.5 = 1990 times per second.

The total number of cycles required for polling is

1990 x 500 = 995,000 cycles per second.

Thus, the total time spent on I/O when using polling with interrupts is
2,400,000 + 995,000 = 3,395,000 cycles per second.

d. The interrupt overhead is 1000 cycles per second for a context switch to the ISR and 1000 cycles per second back from it. This is a total of 2 x 1000 cycles per second. With 200 interrupts per second, this is
200 x 2000 = 400,000 cycles per second.

The polling overhead is 500 cycles per second. Thus, for the same overhead as interrupts, the polling operation should be performed
400,000 / 500 = 800 times per second,
or 1/800 = every 1.25 msec.

Direct Memory Access (DMA)

Direct memory access is a technique, where by the CPU passes its control to the memory subsystem or one of its peripherals, so that a contiguous block of data could be transferred from peripheral device to memory subsystem or from memory subsystem to peripheral device or from one peripheral device to another peripheral device.

Advantage of DMA

The transfer rate is pretty fast and conceptually you could imagine that through disabling the tri-state buffers, the system bus is isolated and a direct connection is established between the I/O subsystem and the memory subsystem and then the CPU is free. It is idle at that time or it could do some other activity. Therefore, the DMA would be quite useful, if a large amount of data needs to be transferred, for example from a hard disk to a printer or we could fill up the buffer of a printer in a pretty short time.

As compared to interrupt driven I/O or the programmed I/O, DMA would be much faster.

What is the consequence? The consequence is that we need to have another chip, which is a **DMA controller.** "A DMA controller could be a CPU in itself and it could control the total activity and synchronize the transfer of data". DMA could be considered as a technique of transferring data from I/O to memory and from memory to I/O without the intervention of the CPU. The CPU just sets up an I/O module or a memory subsystem, so that it passes control and the data could be passed on from I/O to memory or from memory to I/O or within the memory from one subsystem to another subsystem without interaction of the CPU. After this data transfer is complete, the control is passed from I/O back to the CPU.

Now we can illustrate further the advantage of DMA using following example.

Example of DMA

If we write instruction load as follows:

```
load [2], [9]
```

This instruction is illegal and not available in the SRC processor. The symbols [2] and [9] represent memory locations. If we want to have this transfer to be done then two steps would be required. The instruction would be:

```
load r1, [9]
store r1, [2]
```

Thus it is not possible to transfer from one memory location to another without involving the CPU. The same applies to transfer between memory and peripherals connected to I/O ports. For example we cannot have:

```
out [6], datap
```

It has to be done again in two steps:

```
load r1, [6]
out r1, datap
```

Similar comments apply to the "in" instruction. Thus the real cause of the limited transfer rate is the CPU itself. It acts as an unnecessary middle man. The example illustrates that in general, every data word travels over the system bus twice and this is not necessary, and therefore, the DMA in such cases is pretty useful.

DMA Approach

The DMA approach is to turn off i.e. through tri-state buffers and therefore, electrically disconnect from the system bus, the CPU and let a peripheral device or a memory subsystem or

any other module or another block of the same module communicate directly with the memory or with another peripheral device. This would have the advantage of having higher transfer rates which could approach that of limited by the memory itself.

Disadvantage of DMA

The disadvantage however, would be that an additional DMA controller would be required, that could make the system a bit more complex and expensive. Generally, the DMA requests have priority over all other bus activities including interrupts. No interrupts may be recognized during a DMA cycle.