COMPUTER ORGANIZATION (IS F242)

LECT 09_10: INTERRUPTS, MULTIPLICATION

User Visible Registers

- General Purpose
  - Between 8 – 32 (Fewer = more memory references)
- Data Registers
  - To hold the data
- Address Registers
  - Devoted to a particular addressing mode
  - Ex: Segment Registers, Index Registers, Stack Pointer
- Condition Codes
  - Sets of individual bits
    - e.g. result of last operation was zero
  - Can be read (implicitly) by programs
    - e.g. Jump if zero
  - Can not (usually) be set by programs

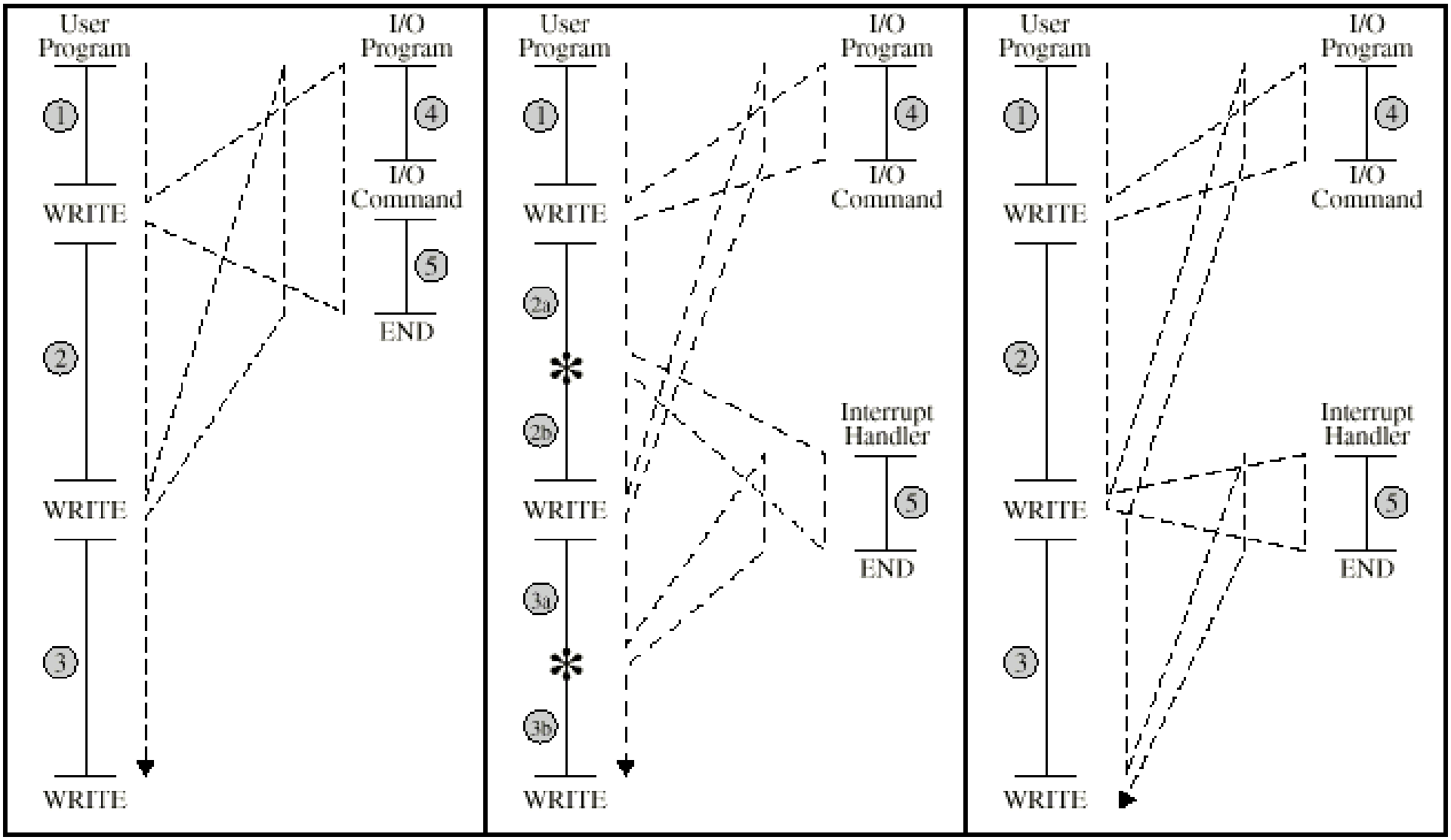
Condition codes

- IA-64 and MIPS do not use condition codes
  - They specify condition branch instructions.
  - Compares the values and act on the result
- Advantages of condition codes
  - Can reduce the number of COMPARE and TEST instructions needed
  - BRANCH is simplified relative to composite instructions such as TEST AND BRANCH
  - Condition codes facilitate multiway branches
- Disadvantages
  - Condition codes add complexity (hardware & software)
  - Condition codes are irregular (not a part of main datapath)
  - Needs additional instructions like bit checking, loop control

Interrupts

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Program
  - e.g. overflow, division by zero
- Timer
  - Generated by internal processor timer
  - Used in pre-emptive multi-tasking
- I/O
  - from I/O controller
- Hardware failure
  - e.g. memory parity error

Program Flow Control



(a) No interrupts

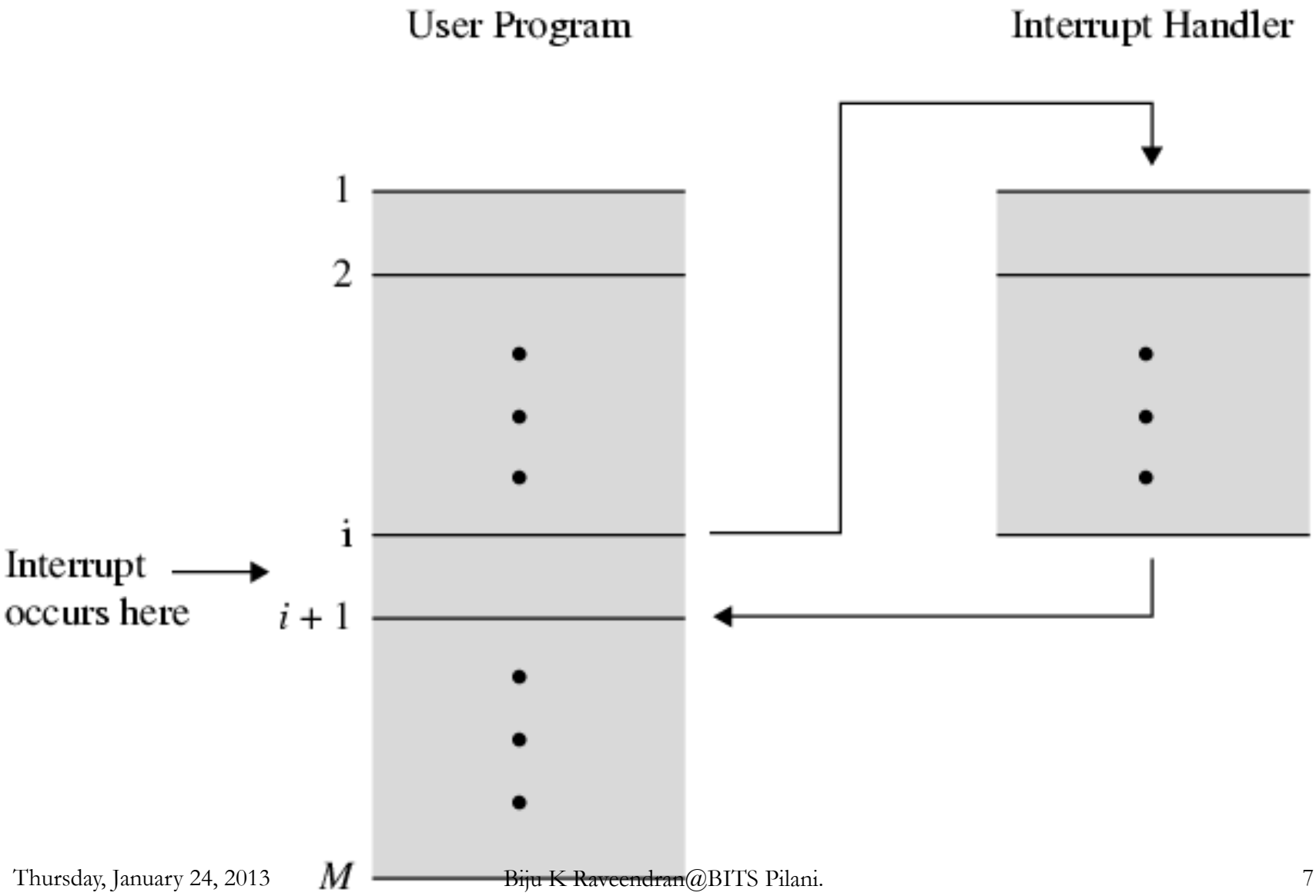
(b) Interrupts; short I/O wait

(c) Interrupts; long I/O wait

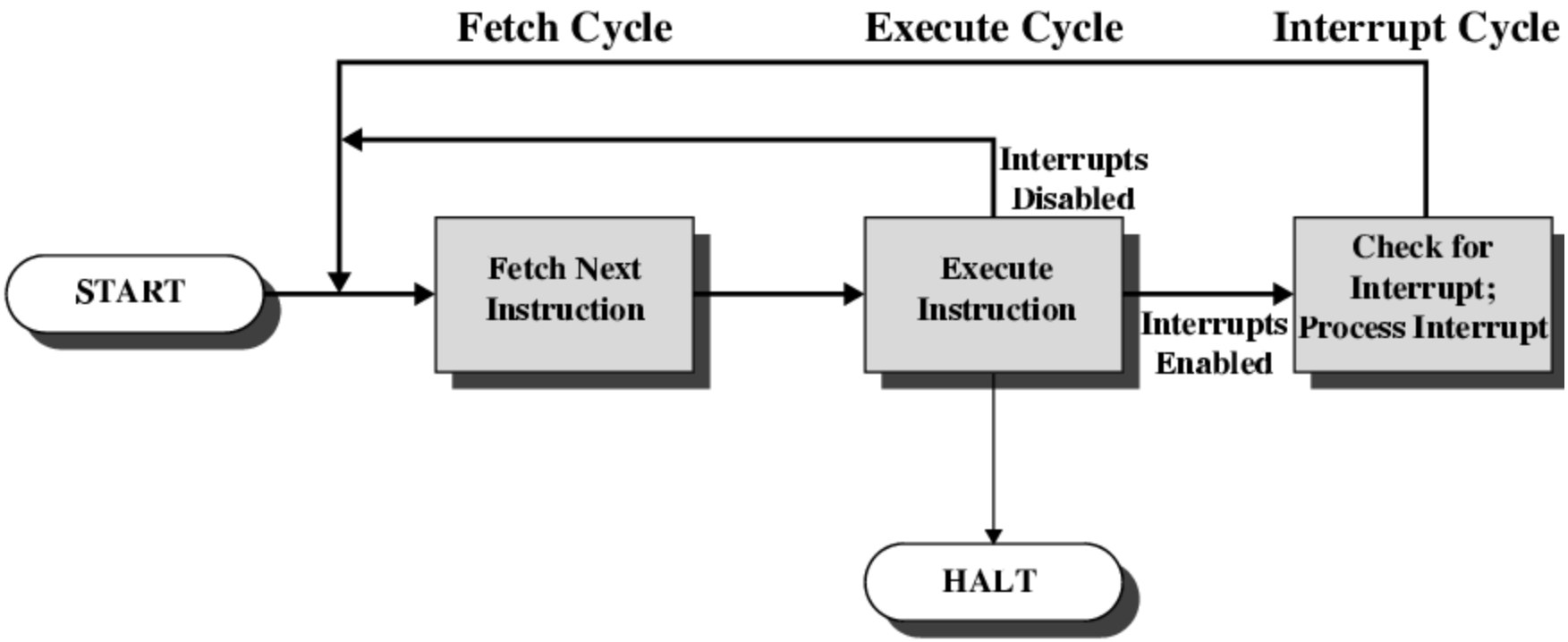
Interrupt Cycle

- Added to instruction cycle
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

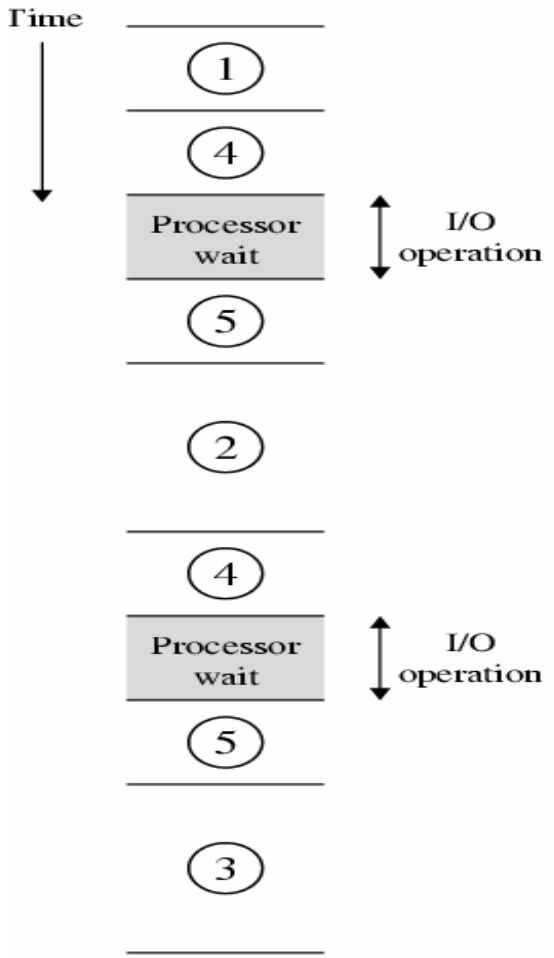
Transfer of Control via Interrupts



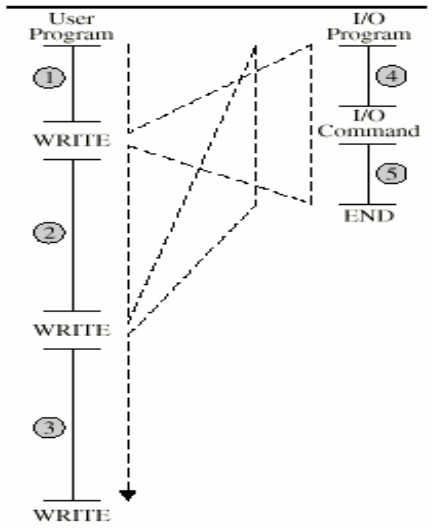
Instruction Cycle with Interrupts



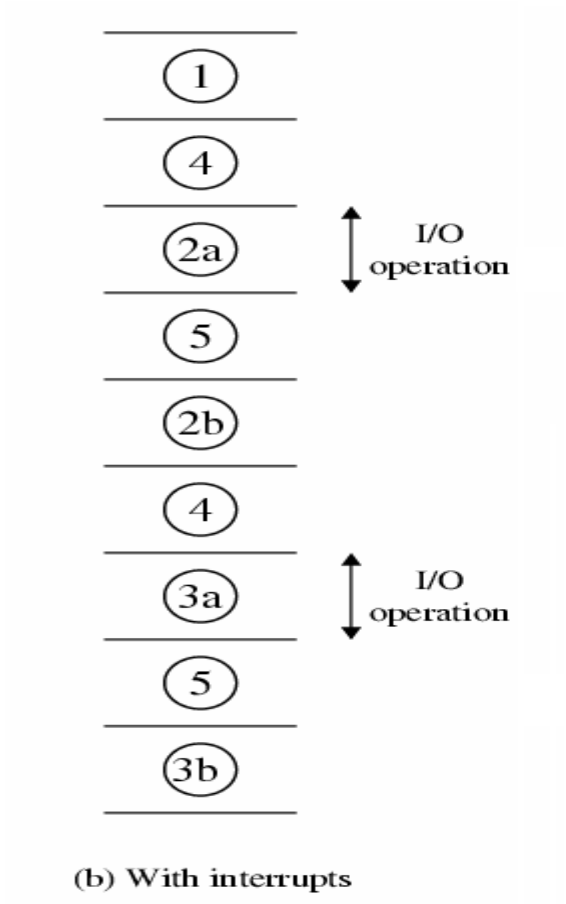
Without Interrupt



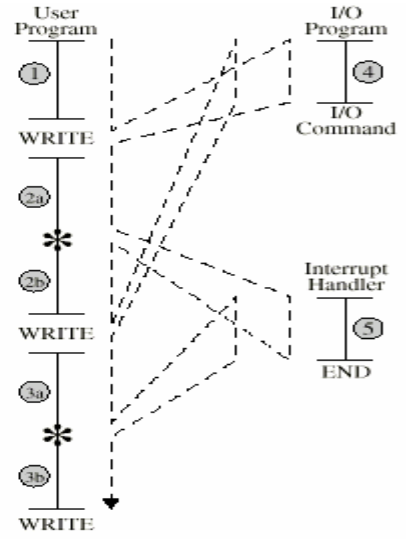
(a) Without interrupts



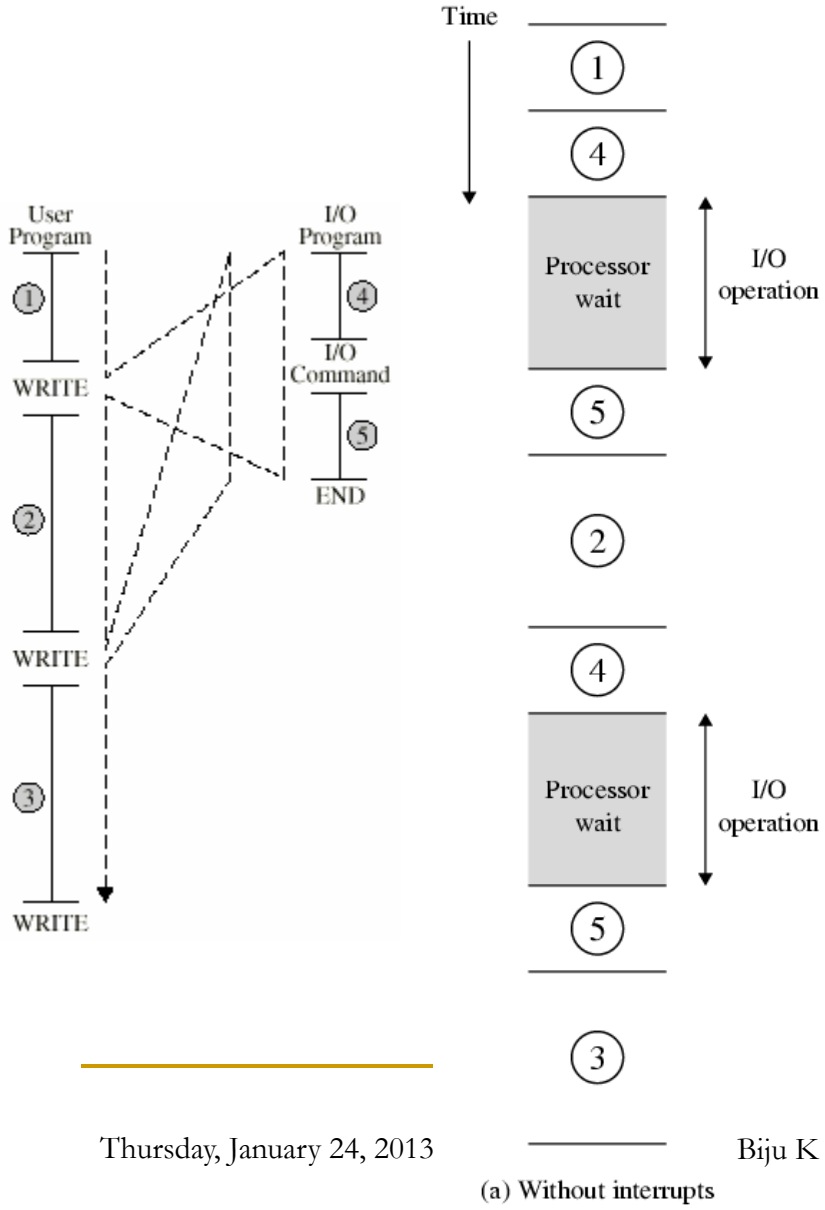
With interrupt



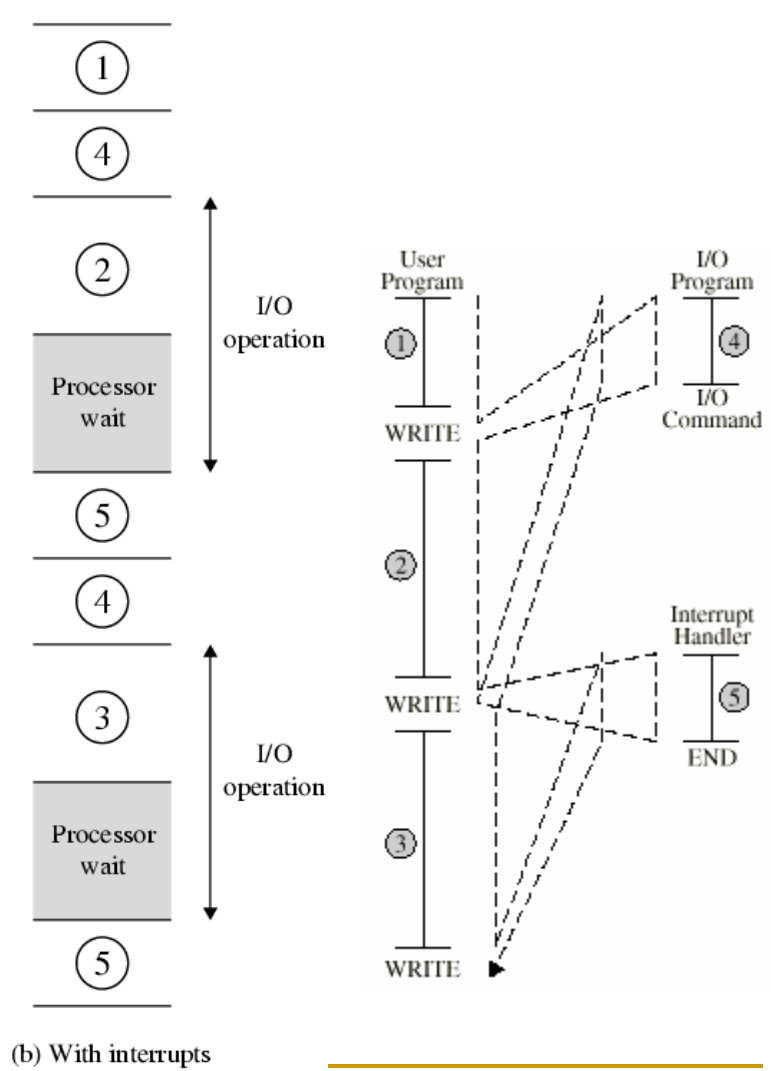
(b) With interrupts



Program Timing Long I/O Wait

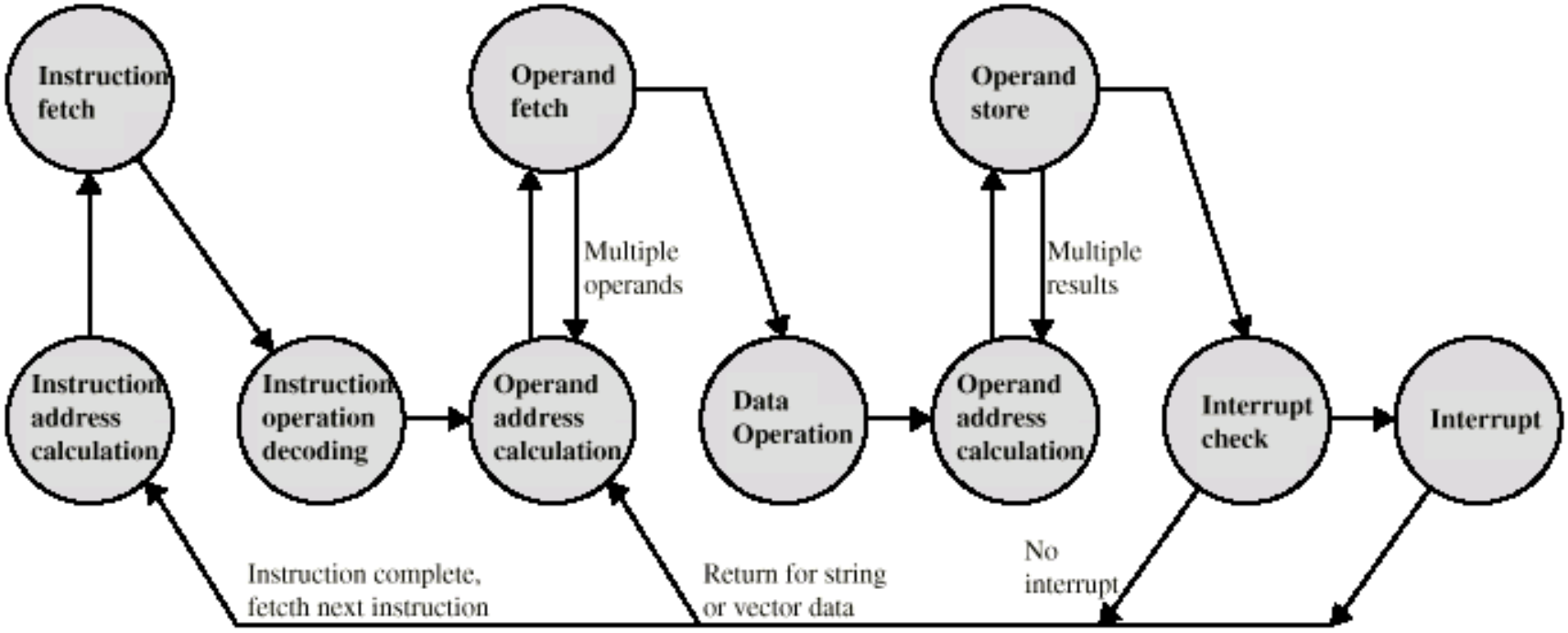


(a) Without interrupts



(b) With interrupts

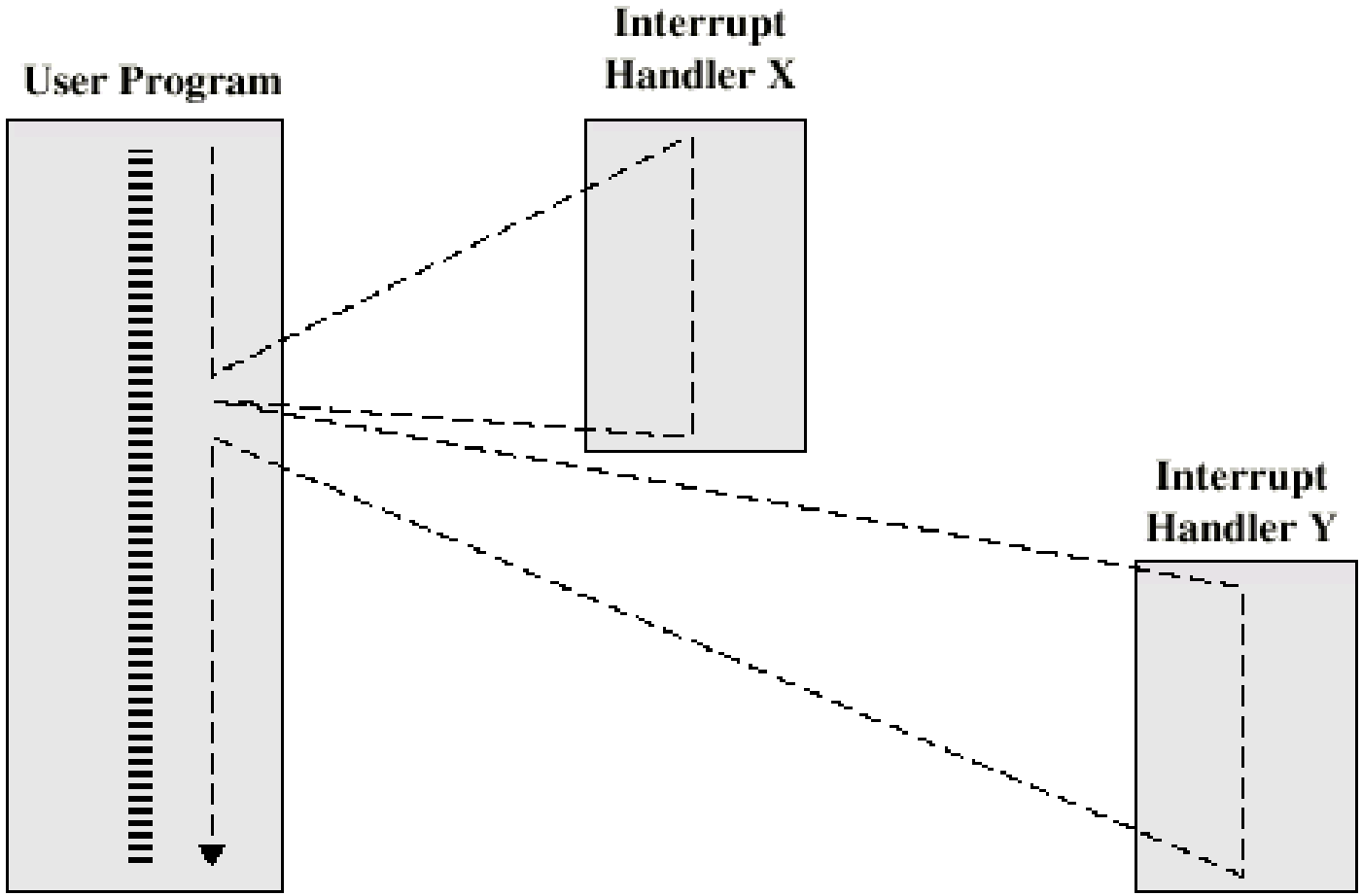
Instruction Cycle (with Interrupts) - State Diagram



Multiple Interrupts

- Disable interrupts
  - Processor will ignore further interrupts while processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur

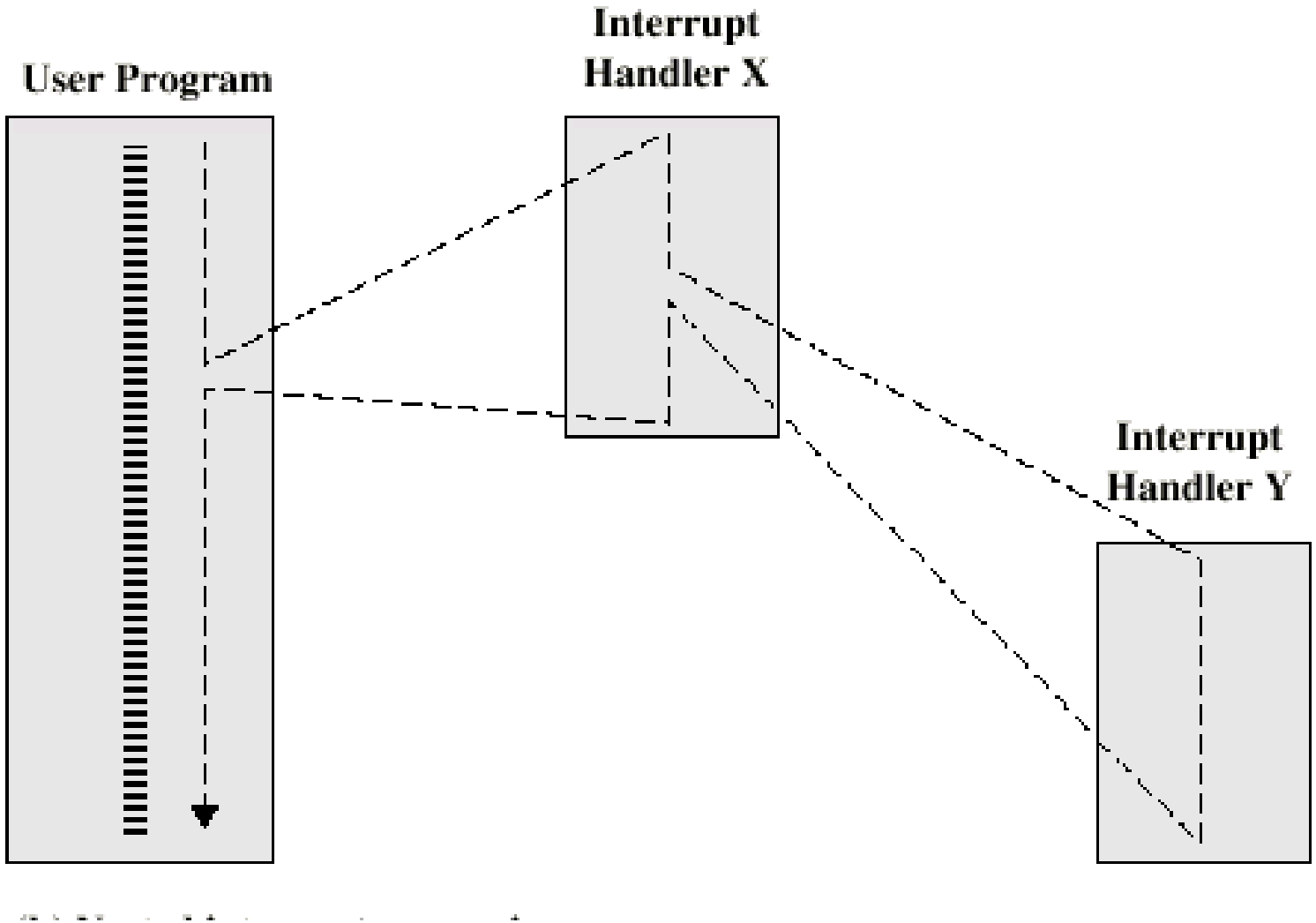
Multiple Interrupts - Sequential



Multiple Interrupts

- Define priorities
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

Multiple Interrupts – Nested

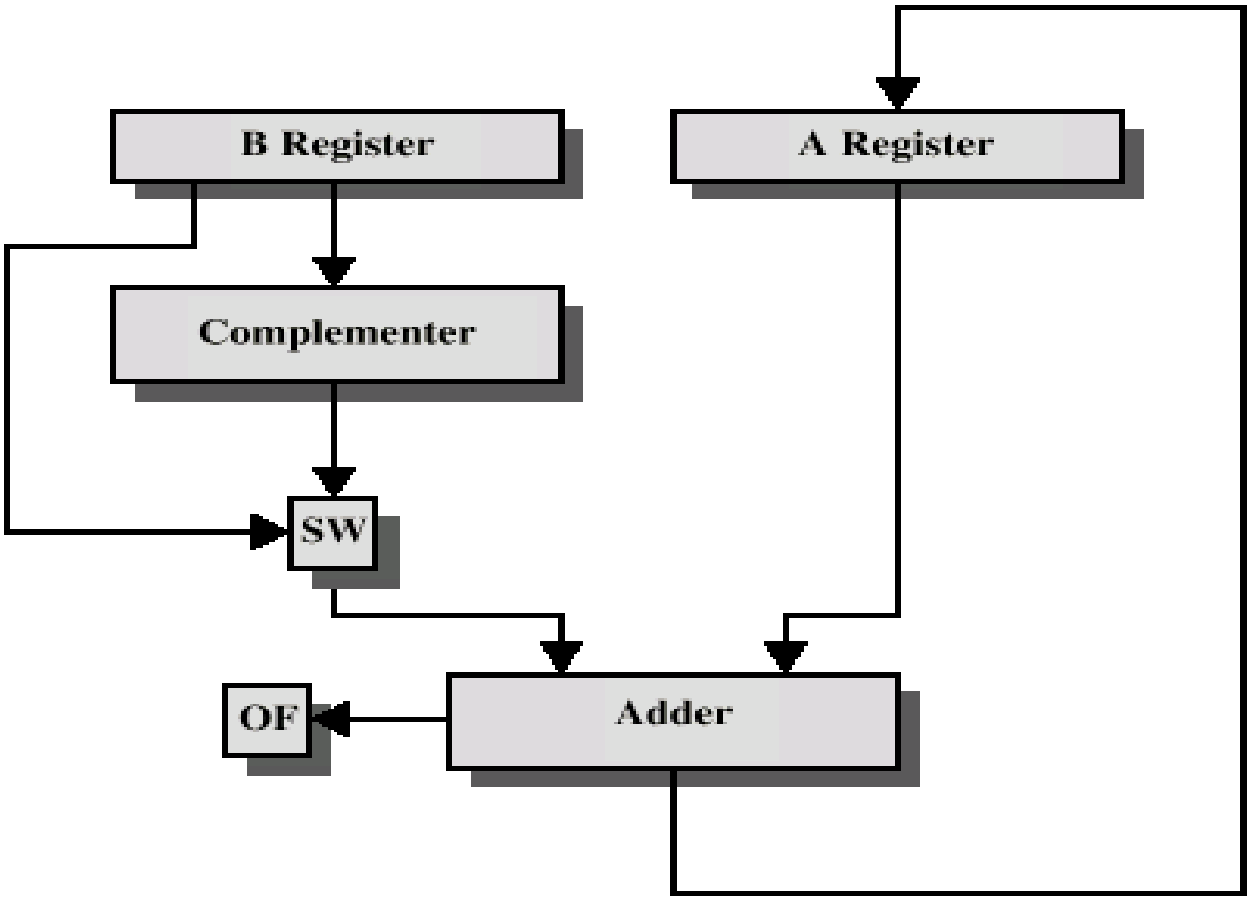


Multiplication & Division

Addition and Subtraction

- Normal binary addition
- Monitor sign bit for *overflow*
- Subtraction
  - a - b = a + (-b)
- Hardware Implementation is simple
  - Addition and complement circuits required

Hardware for Addition and Subtraction

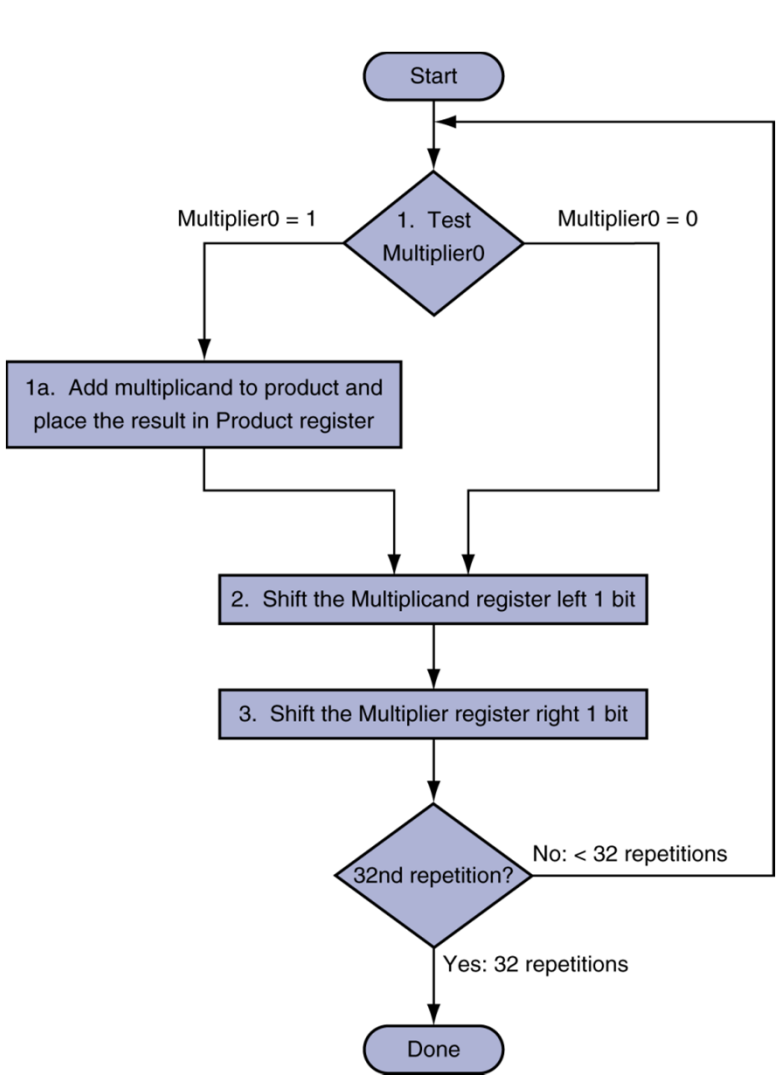


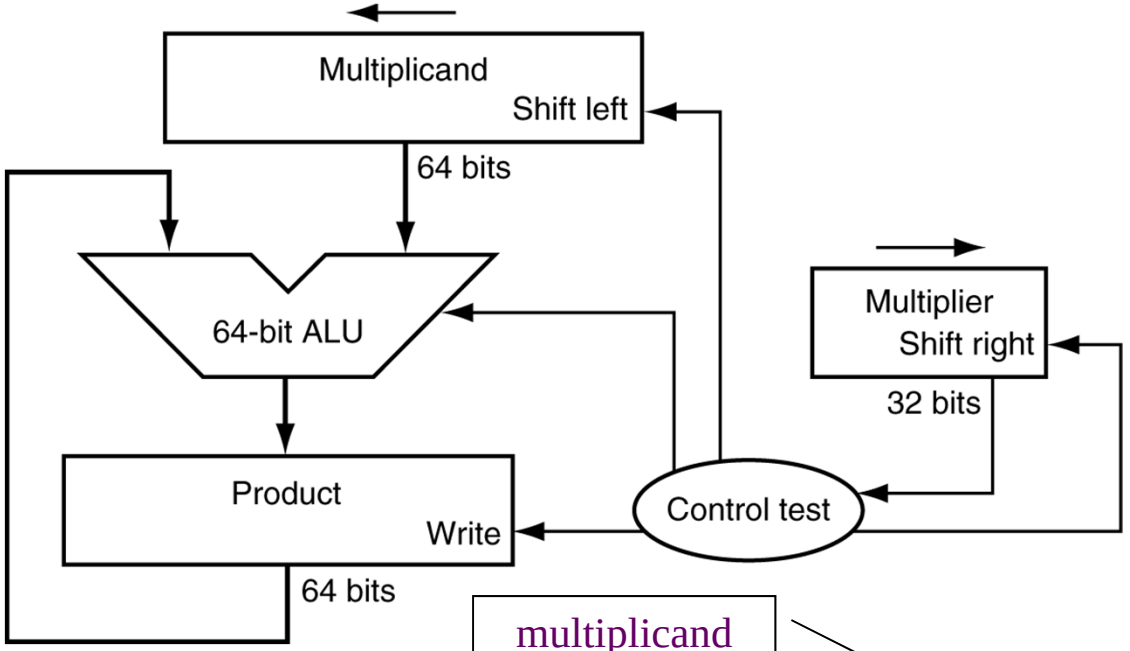
OF = overflow bit
SW = Switch (select addition or subtraction)

Multiplication

- Complex
- Work out partial product for each digit
  - (1000 X 1001 = 1001000)
- Take care with place value (column)
- Add partial products
- What are changes required to do in the above manual approach for computerization???
  - Running addition on the partial products
  - Few registers are required
  - For each 1 in the multiplier an add and shift operation is required
  - For 0 only shift operation is required

Multiplication Hardware





multiplicand

multiplier

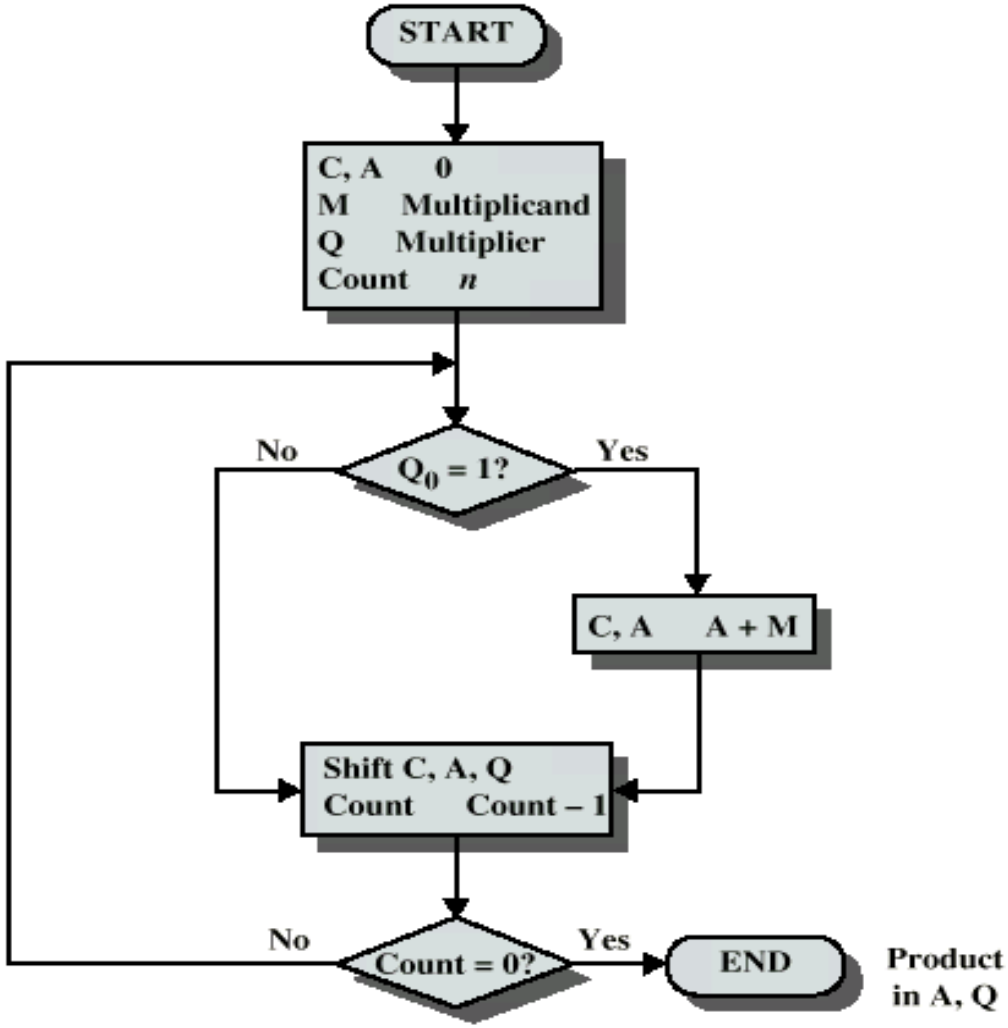
product

```
   1000
×  1001
------
   1000
  0000
 0000
1000
-------
1001000
```

Example

$$
\begin{array}{r}
1011 \\
\times\ 1101 \\
\hline
1011 \\
0000 \\
1011 \\
1011 \\
\hline
10001111
\end{array}
$$

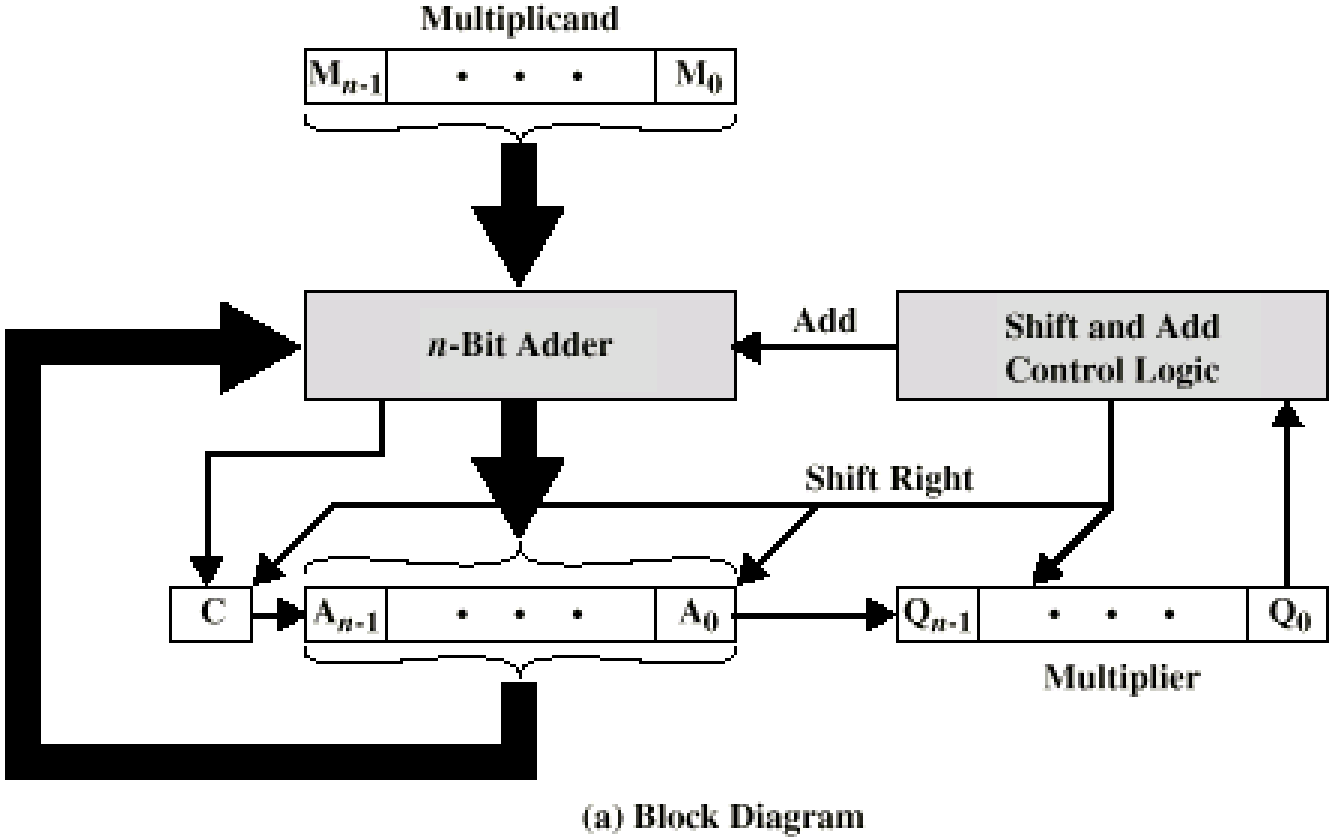
Flowchart for Unsigned Binary Multiplication



Execution of Example

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

Unsigned Binary Multiplication- Optimized



(a) Block Diagram

- Perform steps in parallel: add/shift
- One cycle per partial-product addition
- That's ok, if frequency of multiplications is low

Multiplying Negative Numbers

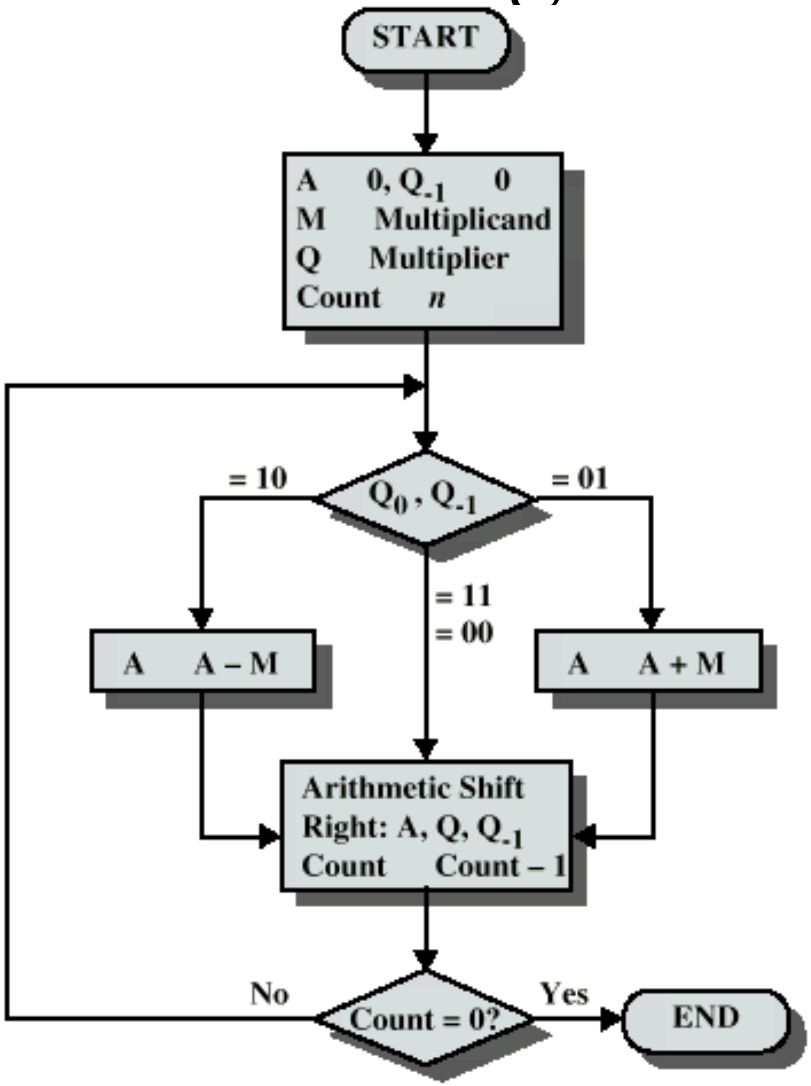
- Solution 1
  - Convert to positive numbers
  - Multiply as unsigned method
  - If signs were different, take 2's complement of the result
- Solution 2
  - Booth's Algorithm

Booth's Algorithm Principle

- Consider the following cases:
  - Mx(00011110)
  - Mx(01111010)
  - Mx(11111010)

- Conclusion
  - Strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^n to 2^k can be treated as $2^{n+1} – 2^k$

Booth's Algorithm



Example of Booth's Algorithm

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A ← A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A ← A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

A	Q	Q-1	M	Count	
00000	**10111**	**0**	**10011**	**101**	**Initial Values**
01101			Subtract M		
01101					
00110	11011	1	Ashift	100	
00011	01101	1	Ashift	011	
00001	10110	1	Ashift	010	
10011			Add M		
10100					
11010	01011	0	Ashift	001	
01101			Subtract M		
00111					
00011	***10101***	1	Ashift	000	

Final result is in A and Q register i.e. **0001110101**