CPU

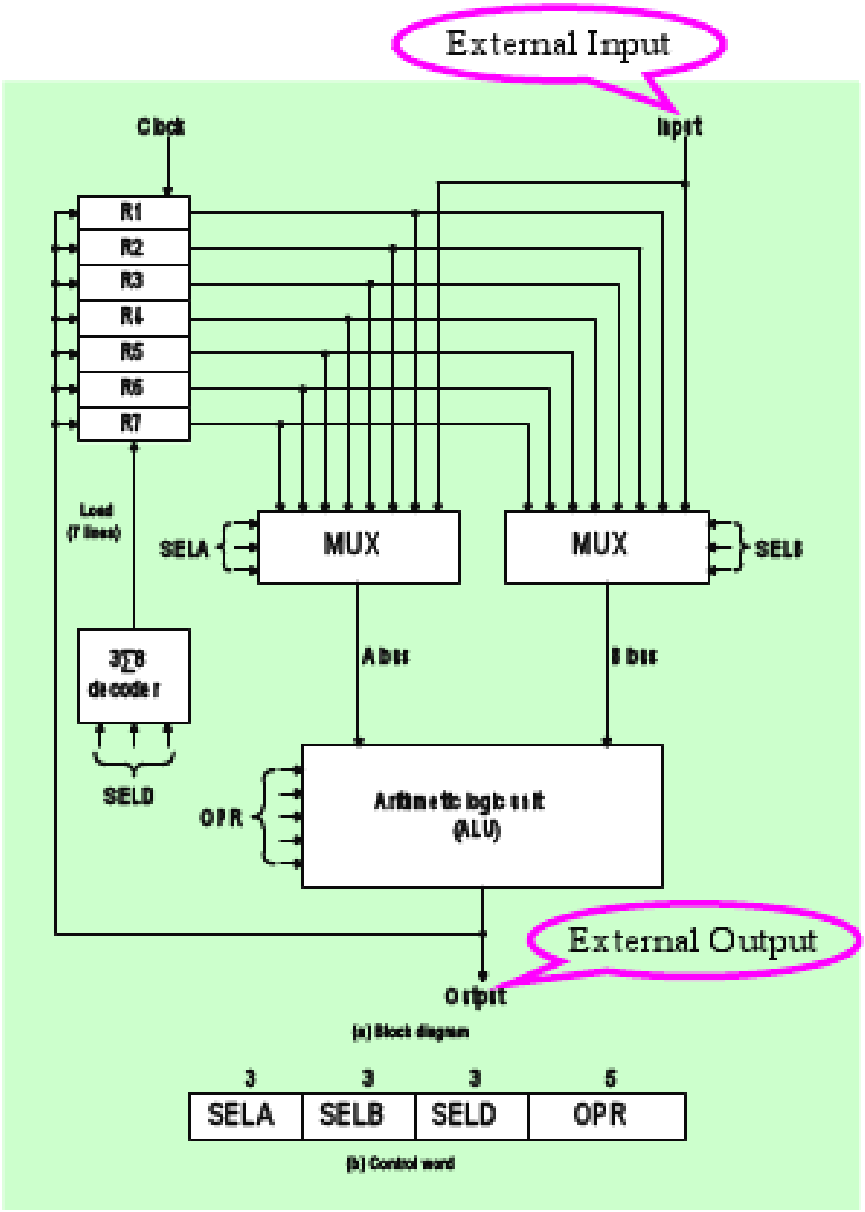
MIT C 201
Computer Organization & Architecture

Department of Information Technology
Central University of Kashmir

Chapter 8: Central Processing Unit

- Three major parts of CPU
  - Register set
  - ALU
  - Control
- Design examples of simple CPU
  - Hardwired (Chapter 5)
  - Microprogrammed (chapter 7)
- Chapter 8
  - Organization and architecture of CPU with emphasis on the users view of the computer
  - A user who programs the computer in machine/assembly language must be aware of
    - Instruction formats
    - Addressing modes
    - Register sets
- Last section presents the concept of RISC

- **General Register organization**
  - Register
    - Memory locations are needed for storing pointers, counters, return address, temporary results, partial products during multiplication
    - Memory access is the most time consuming operation in a computer
    - More efficient way is to store intermediate values in processor registers
- **Bus Organization for Seven CPU Registers**
  - 2 **MUX**: select one of seven registers or external data input SELA and SELB
  - **BUS A** and **BUS B** form the common inputs to a common ALU
  - **ALU:** OPR determines the arithmetic and logic microoperation
    - The result of the microoperation is avaliable for external data output and also goes into the inputs of all the registers
  - **3 * 8 Decoder:** Select the register (By SELD) that receives the information from ALU



- Binary Selector Input
  - 1) MUX A Selector (SELA): to place the contents of R2 into BUS A
  - 2) MUX B Selector (SELB): to place the contents of R3 in BUS B
  - ALU operation selector (OPR): to provide arithmetic addition R2 + R3
  - Decoder Selector (SELD): to transfer the content of output BUS into R1
- Control Word
  - 14 bit control word (4 Fields)
    - SELA(3 bits): select the source register for the A input of the ALU
    - SELB (3 bits): select the source register for the B input of the ALU
    - SELD(3 bits): select destination register using 3 * 8 Decoder
    - OPR (5 bits): select one of the operations in the ALU

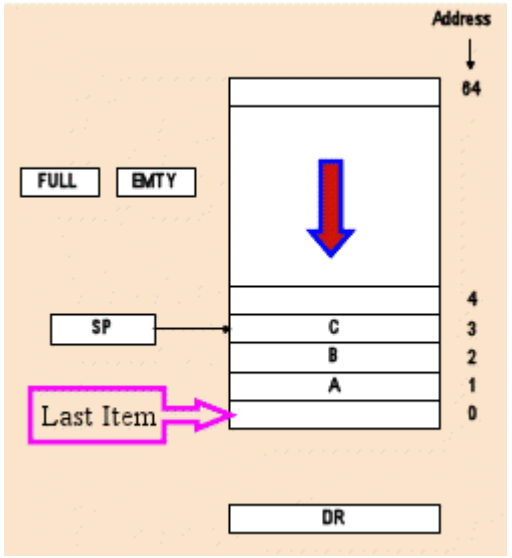
    Encoding of register selection fields:
    - SELA or SELB =000 (input): MUX selects the external input data
    - SELD =000 (None): no destination register is selected but the contents of the output bus are avaliable in the external output

    Encoding of ALU operation (Tab 8.2)
- Examples of Micro operations

  TSFA(Transfer A): R7<- R1, External output <-R2, External output<-external input

  XOR: R5<- 0(XOR R5 R5)

- Stack Organization
- Stack or LIFO (last in- first out)
  - A storage device that stores information
  - Item stored last is the first item retrieved= a stack of trys
  - Stack pointer (SP)
    - The register that holds the address of the stack
    - SP always points at the top item of the stack
  - Operations: insertion and deletion of items
    - PUSH: Push down=insertion
    - POP: Pop up = deletion
  - Stack:
    - 1) register stack
    - 2) memory stack



- Register Stack
  - Push: SP<-SP+1
    M[SP]<-DR
    If (sp=0) then (FULL <-1) //mark not empty
    EMTY <- 0

    The first item is stored at address 1 and last item is stored at address 0
  - POP: DR <- M[SP] : Read item from the top of the stack
    SP<-SP -1 : Decrement Stack pointer
    If (Sp=0) then (Empty <- 1) : Check if stack is empty
    Full <- 0 : Mark not full

- Memory Stack
  - Push: SP<-SP-1
    M[SP]<-DR
  - POP: DR <- M[SP]
    SP<-SP +1

•Error Condition
•PUSH when FULL =1
•POP when empty = 1

- Stack Limits

Check for the Stack overflow (full) / underflow (empty)

Checked by using two registers

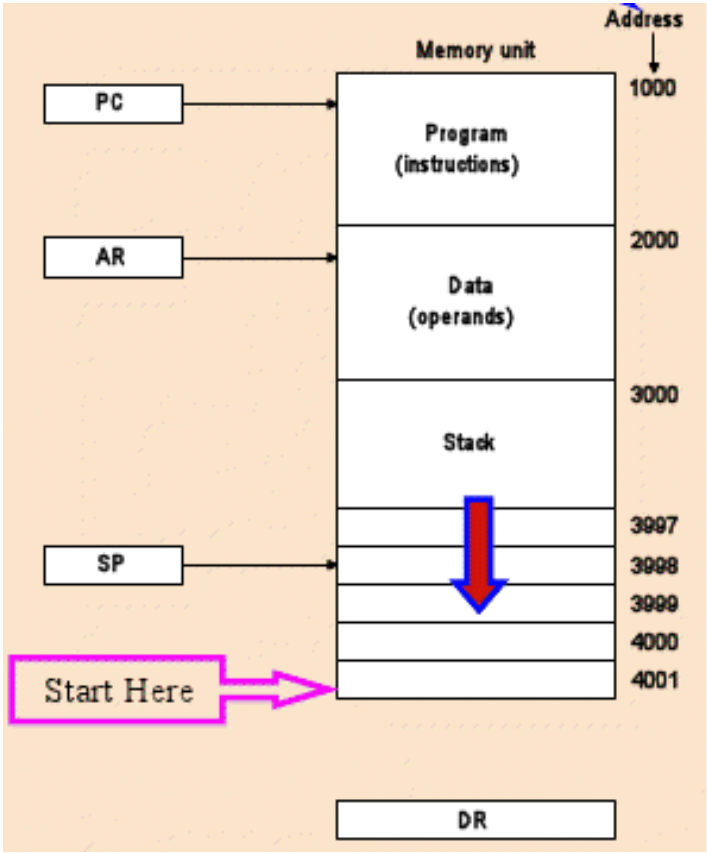
Upper limit and Lower Limit registers

After PUSH operation

SP compared with the upper limit register

After POP operation

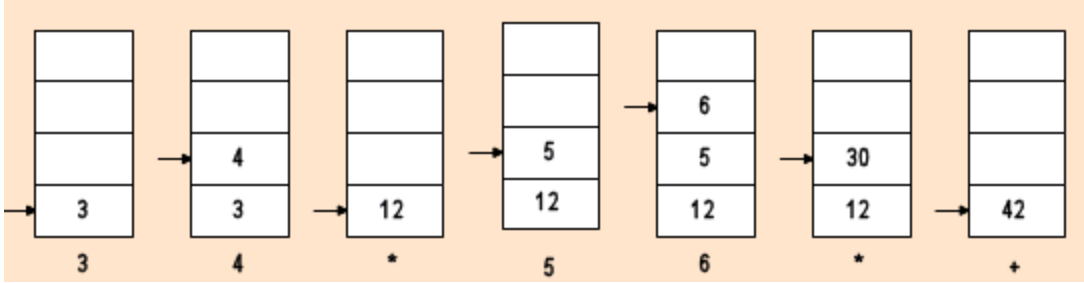
SP compared with the lower limit register



- Reverse Polish Notation (RPN)
  - The common mathematical method of writing arithmetic expressions imposes dificulties when evaluated by a computer
  - A stack organization is very effective for evaluating arithmetic expressions

A * B + C * D → AB*CD*+

(3 *4) + (5 * 6) → 34* 56* +



- Instruction Formats
  - Fields in Instruction Formats
    - Operation Code field
    - Address field
    - Mode field
- 3 Types of CPU Organizations

Single AC	ADD X	AC ← AC + M[X}
General Register Organization	ADD R1, R2, R3	R1 ← R2 + R3
Stack Organization	PUSH X	TOS ← M[X}

- Influence of number of addresses on computer instruction

  X= (A+ B) * (C * D)
  - 4 arithmetic operations: ADD SUB MUL DIV
  - 1 transfer operation to and from memory and general register : Mov
  - 2 transfer operations to and from memory and AC register: Store, Load
  - Operand Memory Addresses: A, B, C, D
  - Result Memory Address: X

- # Three Address Instruction

ADD	R1, A, B	R1 ←M[A] + M[B]
ADD	R2,C,D	R2← M[C] + M[D]
MUL	X, R1, R2	M[X] ← R1 * R2

Each address field specifies either a processor register or a memory operand

Short programme

Require too many bits to specify three addresses

- # Two Address Instruction

MOV	R1, A	R1← R1 + M[A]
ADD	R1, B	R1← R1 + M[B}
MOV	R2, C	R2 ← M[C]
ADD	R2, D	R2 ← M[D]
MUL	R1, R2	R1 ← R1 * R2
MOV	X, R1	M[X] ← R1

Most common in commercial computers

Each address field specify either a processor register or a memory operand

- One Address Instruction

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	AC ← M[X]

All operations are done between AC register and memory operand

- Zero Address Instruction

PUSH	A	TOS ← M[A]
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (A+B) * (C+D)
POP	X	M[X] ← TOS

Stack organized computer does not use an address field for the instructions ADD and MUL

PUSH and POP instructions need an address field To specify the oprand

- RISC Instruction
  - Only use LOAD and STORE instruction when communicating between memory and CPU
  - All other instructions are executed within the registers of the CPU without referring to memory

Program to evaluate X = (A+B) * (C+D)

```
LOAD R1, A          R1<- M[A]
LOAD R2, B          R2<-M[B]
LOAD R3, C          R3<-M[C]
LOAD R4, D          R4<- M[D]
ADD R1, R1, R2      R1<-R1+R2
ADD R3, R3, R4      R3<-R3+R4
MUL R1, R1, R3      R1<-R1*R3
STORE X, R1         M[X]<-R1
```

Addressing Modes

- **Addressing Mode**
  - To give programming versatility to user
    - Pointers to memory, counters for loops, indexing of data..
  - To reduce the number of bits in the addressing field of the instruction

- **Instruction Cycle**
  - Fetch the instruction from memory and PC<- PC + 1
  - Decode the instruction
  - Execute the instruction

- **Program Counter**
  - Program counter keeps track of the instructions in the program stored in memory
  - PC holds the address of the instruction to be executed next
  - PC is incremented each time an instruction is fetched from memory

- **Addressing mode of the instruction**
  - Distinct Binary code
  - Single binary code
- Instruction format with mode field

- **Implied Mode**
  - Operands are specified implicitly in the definition of instruction
  - Examples
    - **COM**: Complement Accumulator
      - Operand AC is implied in the definition of the instruction
    - **PUSH**: Stack Push
      - Operand is implied to be on top of the stack

- **Immediate mode**
  - Operand field contain the actual operand
  - Useful for initiating registers to a constant value
  - Example: **LD #NBR**

- **Register Mode**
  - Operands are in registers
  - Register is selected from a register field in the instruction
    - K-bit register field can specify any one of 2^K registers
  - Example: **LD R1** AC <-R1

- **Register Indirect Mode**
  - Selected register contains the address of the operand rather than the operand itself.
  - Address field of the instruction uses fewer bits to select a memory address
  - Example: **LD (R1)** AC <- M[R1]

- **Autoincrement or Autodecrement mode**
  - Similar to register indirect mode except that
    - The register is incremented after its value is used to access memory
    - The register is decrement before its value is used to access memory
    - Example (Autoincrement): **LD (R1+)** AC <- M[R1], R1<-R1+1

- **Direct Addressing Mode**
  - Effective address is equal to the address field of the instruction (Operand).
  - Address field specifies the actual branch address of the instruction
  - Example: LD ADR AC<- M[ADR]

- **Indirect Addressing Mode**
  - Address field of instruction gives the address where the effective address is stored in memory
  - Example: LD @ADR AC <- M[M[ADR]]

- **Relative addressing Mode**
  - PC is added to the address part of the instruction to obtain the effective address
  - Example: LD $ADR AC<- [PC+ADR]

- **Indexed Addressing mode**
  - XR (index register) is added to the address part of the instruction to obtain effective address
  - Example: LD ADR(XR) AC<-[ADR+XR]

- **Base register Addressing mode**
  - The content of the base register is added to the address part of the instruction to obtain the effective address

- Similar to indexed addressing mode except that the register is now called base register instead of index register
  - Index register (XR): LD ADR(XR) AC<-M[ADR+XR]
    - Index register hold an index number that is relative to the address part of the instruction
  - Base register (BR): LD ADR(BR) AC<- M[BR+ADR]
    - Base register hold a base address
    - Address part of the instruction gives the displacement relative to the base address.

Example

Numerical Example

Addressing Mode	Effective Address	Content of AC
Immediate Address Mode	201	500
Direct Address Mode	500	800
Indirect Address Mode	800	300
Register Mode		400
Register Indirect Mode	400	700
Relative Address Mode	702	325
Indexed Address Mode	600	900
Autoincrement Mode	400	700
Autodecrement Mode	399	450

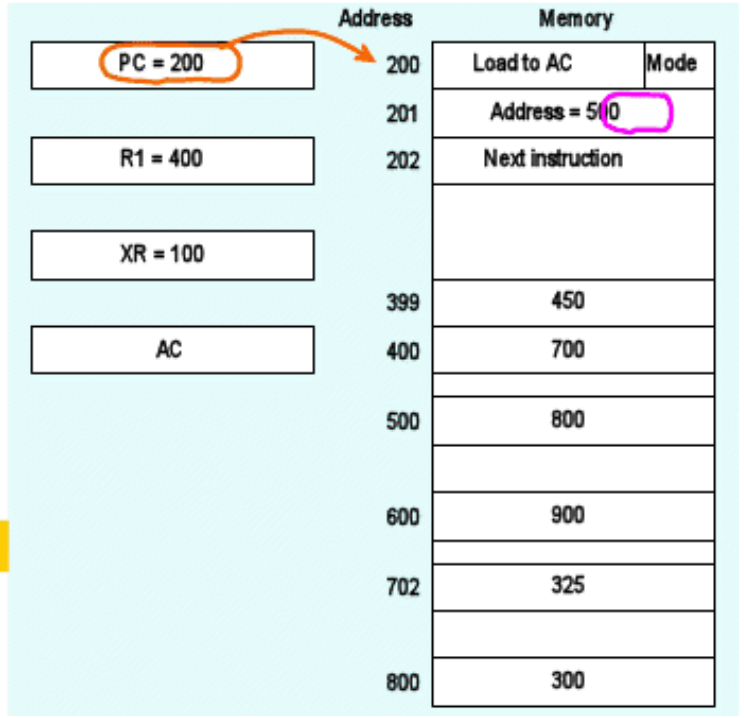
R1 = 400

500 + 202 (PC)

500 + 100 (XR)

R1 = 400 (after)

R1 = 400 -1 (prior)



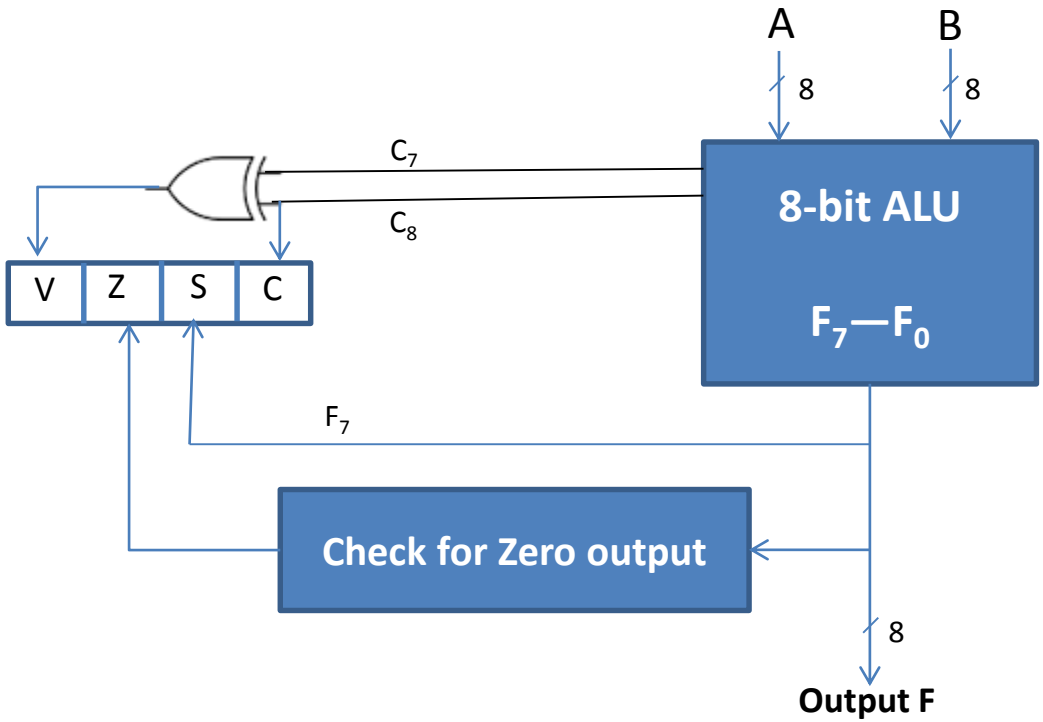
Data Transfer and Manipulation

- Most computer Instructions can be classified in to three categories:
  - Data Transfer
  - Data Manipulation
  - Program control Instructions

  The instruction set of a particular programmer determines the register transfer operations and control decisions that are available to the user.

- **Data Transfer Instructions**
  - Typical Data Transfer instructions
    - **Load**: Transfer from memory to the processor register, usually an AC (memory read)
    - **Store**: transfer from processor register to Memory (memory write)
    - **Move:** transfer from one register to another register
    - **Exchange:** swap information between two registers or a register and a memory
    - I**nput/output**: transfer information among processor registers and input/output devices
    - **Push/POP:** transfer data between processor registers and a memory stack
- Different addressing modes for Load Instruction
  - @: indirect Address
  - $: Address relative to PC
  - #: immediate mode
  - (): index mode

- **Data Manipulation Instruction**
  - Arithmetic
    - INC, DEC, ADD, SUB, MUL, DIV, ADDC, NEG(2's compliment)
  - Logic & bit manipulation
    - CLR, COM,AND, OR, XOR, CLRC, SETC, COMC, EI, DI
  - Shift instruction
    - SHR, SHL,SHRA, SHLA, ROR, ROL, RORC, ROLC
- **Program Control**
  - Program Control Instructions
    - BR, JMP, SKP, CALL, RET,
  - Status bit conditions
    - To check different conditions for branching instructions like CMP (compare) or TEST (by ANDing) can be used.
    - Certain status bit conditions are set as a result of these operations

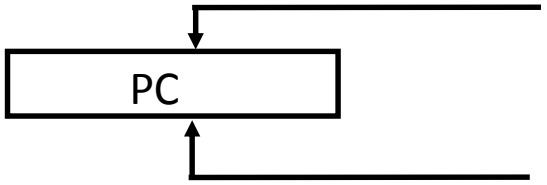


- 4-bit status register
  - Bit C (carry): set to 1 if the end carry C8 is 1
  - Bit S (Sign): set to 1 if F7 is 1
  - Bit Z (zero): set to 1 if the output of the ALU contains all zeros
  - Bit V (overflow): set to 1 if the exclusive –OR of the last two caries (C8 and C7) is equal to 1
    - Flag example: A-B = A + (2's Comp. Of B): A=11110000, B=00010100

```
 11110000
+11101100
----------
1 1101110
```

C=1, S=1, V=0, Z=0

PROGRAM CONTROL INSTRUCTIONS



+1

In-Line Sequencing (Next instruction is fetched from the next adjacent location in the memory)

Address from other source; Current Instruction, Stack, etc; Branch, Conditional Branch, Subroutine, etc

- Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by –)	CMP
Test(by AND)	TST

* CMP and TST instructions do not retain their results of operations (– and AND, respectively). They only set or clear certain Flags.

CONDITIONAL BRANCH INSTRUCTIONS

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	A = B
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions (A - B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	A < B
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	A = B
BNE	Branch if not equal	$A \neq B$

SUBROUTINE CALL AND RETURN

- Subroutine Call

  Call subroutine
  Jump to subroutine
  Branch to subroutine
  Branch and save return address

- Two Most Important Operations are Implied;

  * Branch to the beginning of the Subroutine
    - Same as the Branch or Conditional Branch

  * Save the Return Address to get the address of the location in the Calling Program upon exit from the Subroutine

- Locations for storing Return Address
  - Fixed Location in the subroutine (Memory)
  - Fixed Location in memory
  - In a processor Register
  - In memory *stack*
    - most efficient way

CALL

$SP \leftarrow SP - 1$
$M[SP] \leftarrow PC$
$PC \leftarrow EA$

RTN

$PC \leftarrow M[SP]$
$SP \leftarrow SP + 1$

PROGRAM INTERRUPT

Types of Interrupts

External interrupts

External Interrupts initiated from the outside of CPU and Memory
- I/O Device → Data transfer request or Data transfer complete
- Timing Device → Timeout
- Power Failure
- Operator

Internal interrupts (traps)

Internal Interrupts are caused by the currently running program
- Register, Stack Overflow
- Divide by zero
- OP-code Violation
- Protection Violation

Software Interrupts

Both External and Internal Interrupts are initiated by the computer HW.
Software Interrupts are initiated by the executing an instruction.
- Supervisor Call → Switching from a user mode to the supervisor mode
  → Allows to execute a certain class of operations
  which are not allowed in the user mode

INTERRUPT PROCEDURE

Interrupt Procedure and Subroutine Call

- The interrupt is usually initiated by an internal or an external signal rather than from the execution of an instruction (except for the software interrupt)

- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction

- An interrupt procedure usually stores all the information necessary to define the state of CPU rather than storing only the PC.

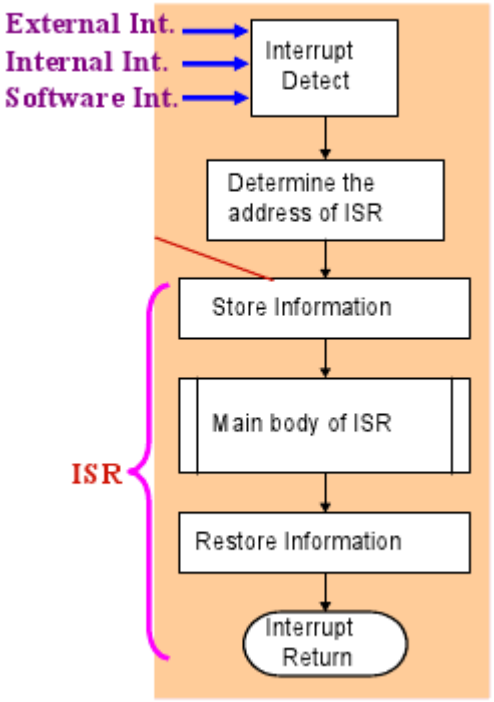
  The state of the CPU is determined from;
  - Content of the PC
  - Content of all processor registers
  - Content of status bits

  Many ways of saving the CPU state depending on the CPU architectures

- Programme Interrupt
  - Program interrupt
    - Transfer program control from a currently running program as a result of an external or internal generated request
    - Control returns to the original program after the service program is executed
  - Interrupt service program V/S Subroutine call
    - An interrupt is initiated by an internal or external signal (except for software interrupt)
      - A subroutine call is initiated from the execution of the instruction (CALL)
    - The address of the interrupt service program is determined by the hardware
      - The address of the subroutine call is determined from the address field of an instruction
    - An interrupt procedure stores all the information necessary to define the state of the CPU
      - A subroutine call stores only program counter (Return address)

- Program Status Word (PSW)
  - Collection of all status bit conditions in the CPU
- Two CPU Operating modes
  - Supervisor (System) mode : privileged instruction
    - When CPU is executing a program which is part of the operating system.
  - User mode: user program

- Types of Interrupts
  - 1) External
    - Come from I/O device , from a circuit monitoring the power supply or from any other external source.
  - 2) Internal (Traps)
    - Caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, protection violation
  - 3) software
    - Initiated by executing an instruction (INT or RST)
    - Used by a programmer to initiate an interrupt procedure at any desired point in the program



- Complex Instruction Set Computer (CISC)
  - Large number of instructions (100 to 250 instructions)
  - Some instructions that perform specialized tasks and are used infrequently
  - A large variety of addressing modes (5 to 20 modes)
  - Variable length instruction formats
  - Instructions that manipulate operands in memory

- Reduced Instruction Set Computer (SISC)
  - Relatively few instructions
  - Relatively few addressing modes
  - Memory access limited to load and store instructions
  - All operations done within the registers of the CPU
  - Fixed-length easily decoded instruction format
  - Single-cycle instruction execution
  - Hardwired rather than micro-programmed control

https://docs.google.com/viewer?url=http://www.qiau.ac.ir/Services/elearning/farokhi/ch08.ppt&chrome=true

Cp9:

http://www.scribd.com/doc/105309127/ch09-morris-mano