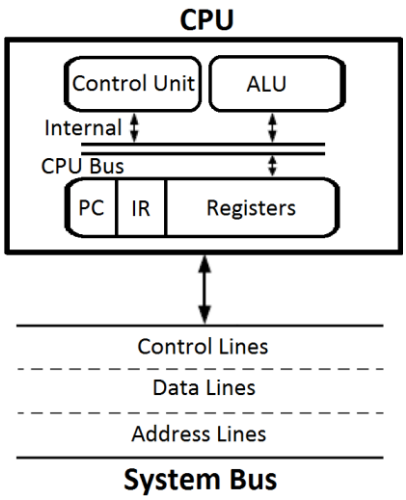
THE CPU

- Stores Intermediate e data during execution of instruction
- Performs the required micro-operations for executing the instructions
- Supervises the transfer of info among the registers and instructs the ALU as to which operation to perform



CPU FUNCTIONS

- CPU functions depend on computers instruction set.
- Computer Architecture = Computer Structure and Behavior as seen by the low level programmer
  - Instruction formats
  - Addressing modes
  - The instruction set
  - The general organization of CPU
- The instruction set provides the specifications for the CPU design and this know how is essential to be able to program efficiently.

Components

Arithmetic logic unit (ALU)	Performs mathematical and logical operations
Control unit (CU)	Decodes the program instruction in the IR, selecting machine resources, such as a data source register and a particular arithmetic operation, and coordinates activation of those resources.

Control and Status Registers

• *Program Counter (PC)*	An incrementing counter that keeps track of the memory address of the instruction that is to be executed next or in other words, holds the address of the instruction to be executed next.
• *Instruction Register (IR)*	A temporary holding ground for the instruction that has just been fetched from memory.
• *Memory Address Register (MAR)*	holds the address of a memory location that is to be read or write
• *Memory Buffer Register (MBR)*	A two-way register that holds data fetched from memory (and ready for the CPU to process) or data waiting to be stored in memory.
• *Program Status Word (PSW)*	Condition Code Flags + other bits defining the status of the CPU (interrupt enabled/disabled, supervisor, etc.)

User Visible Registers

• *General-Purpose Registers*	Some architecture provides a set of registers which can be used without restrictions as operands for any opcode and as address registers

CPU ORGANIZATIONS

- Operations are performed on data stored in memory or processor registers.
- The operands are thus specified by their memory addresses or register address.
- The number of address fields in the instruction format depends on the internal organization of CPU
- In general, most processors are organized in one of 3 ways
    - Single Register (Accumulator) Organization
    - General register organization
    - Stack organization

Single Register (Accumulator) Organization

- Accumulator is the only general purpose register
- Instructions use Zero or One address field.

Advantages

- Accumulator content is meant to be an operand. No requirement for the operand address for one operand. This results in Short instructions and less memory space.
- Instruction cycle takes less time as it saves time in instruction fetching due to absence of operand fetch.

Disadvantages

- Program size increases due to usage of many instructions in complex expressions. Total memory size increases.
- Program execution time increases due to increase in no. of instructions per Program.

General register organization

- Used by most modern computer processors
- Any of the registers can be used as the source or destination for computer operations
- Instructions use 2 or 3 addresses.

Advantages

- Shorter programs than Accumulator Based CPUs
- In Accumulator based CPU, a memory location is required for storing partial results. This additional memory access is avoided here.
- Increase in no. of registers increases CPU efficiency

Disadvantages

- Cost of hardware increases with no. of GPRs.

Stack organization

- All operations are done using the hardware stack
- PUSH and POP instructions which require an address field.
- Other instructions don’t need address fields as the operands are implied to be in the stack.

- For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack

Advantages

- Easy programming
- High compiler efficiency
- Highly suited for block structured languages
- Instructions don't have address field – short instructions.

Disadvantages

- Additional h/w circuitry needed for stack implementation.
- Increased program size.

SOME TRADE OFFS

- A large number of general purpose registers means large number of bits for encoding register operands; specialization of registers reduces this need.
- Too small number of registers creates problems to the programmer and leads to an increased memory traffic.
- The number of general-purpose or data registers is often between 8 - 32.
- RISC processors often have a very large number of registers (~ 100).

EXAMPLE		
(A + B) - (C + D)		
Accumulator Based CPU	**Register Based CPU**	**Stack Based CPU**
LOAD A	LOAD R1, A	PUSH A
ADD B	ADD R1, B	PUSH B
STORE T	LOAD R2, C	ADD
LOAD C	ADD R2, D	PUSH C
ADD D	SUB R1, R2	PUSH D
STORE U	STORE X,R1	ADD
LOAD T		SUB
SUB U		POP X
STORE X		

INSTRUCTION FORMAT

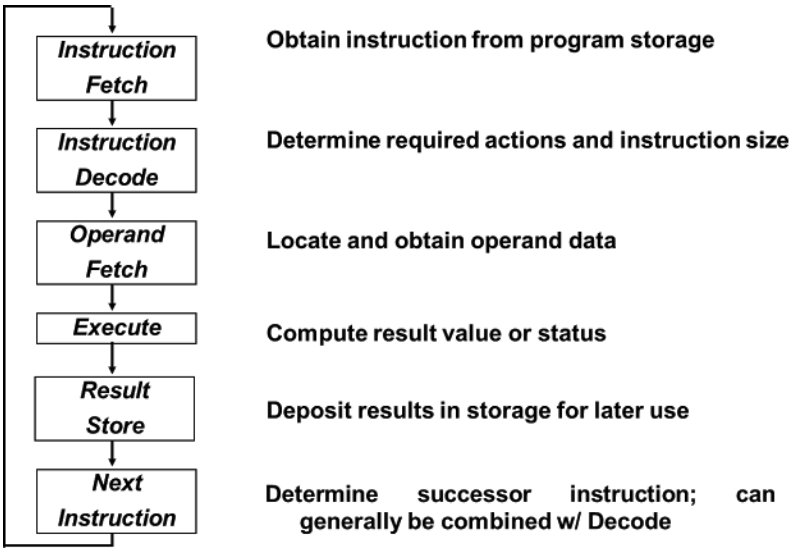
- Instruction Fields
    - OP-code field : Specifies the operation to be performed Group of bits that define various processor operations.
    - Address field : designates memory address(es) or a processor register(s)
    - Mode field : Determines how the address field is to be interpreted (to get effective address or the operand)

Instruction Cycle

An instruction cycle (sometimes called a fetch–decode–execute cycle) is the basic operational process of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction dictates, and carries out those actions. This cycle is repeated continuously by a computer's central processing unit (CPU), from boot-up to when the computer is shut down. Instruction Cycle has mainly 4 phases that are Fetch Phase, Decode Phase, Decision Phase and Execution Phase.

Initiating the cycle

The cycle begins as soon as power is applied to the system, with an initial PC value that is predefined by the system's architecture (for instance, in Intel IA-32 CPUs, the predefined PC value is 0xfffffff0). Typically this address points to a set of instructions in read-only memory (ROM), which begins the process of loading (or *booting*) the operating system.



Flowchart of Instruction Cycle

Fetching the instruction

Step 1 of the Instruction Cycle is called the Fetch Cycle. This step is the same for each instruction:

- The CPU sends PC to the MAR and sends a READ command on the control bus

- In response to the read command (with address equal to PC), the memory returns the data stored at the memory location indicated by PC on the databus
- The CPU copies the data from the data bus into its MDR (also known as MBR, see section Components above)
- A fraction of a second later, the CPU copies the data from the MDR to the Instruction Register (IR)
- The PC is incremented so that it points to the following instruction in memory. This step prepares the CPU for the next cycle.

The Control Unit fetches the instruction's address from the Memory Unit.

Decoding the instruction

Step 2 of the instruction Cycle is called the Decode Cycle. The decoding process allows the CPU to determine what instruction is to be performed, so that the CPU can tell how many operands it needs to fetch in order to perform the instruction. The opcode fetched from the memory is decoded for the next steps and moved to the appropriate registers. The decoding is done by the CPU's Control Unit.

Reading the effective address

Step 3 is evaluating which operation it is. If this is a Memory operation - in this step the computer checks if it's a direct or indirect memory operation:

- **Direct memory instruction** - Nothing is being done.
- **Indirect memory instruction** - The effective address is being read from the memory.

If this is an I/O or Register instruction - the computer checks its kind and executes the instruction.

Executing the instruction

Step 4 of the Instruction Cycle is the Execute Cycle. Here, the function of the instruction is performed. If the instruction involves arithmetic or logic, the Arithmetic Logic Unit is utilized. This is the only stage of the instruction cycle that is useful from the perspective of the end user. Everything else is overhead required to make the execute phase happen.

Three-Address Instructions (Opcode, Destination and Source)

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

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)

- Results in short programs
- Instruction becomes long (many bits)

Two-Address Instructions

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

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

One-Address Instructions

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D)

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	(M[X] ← AC)

Zero-Address Instructions

Can be found in a stack-organized computer

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

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

REVERSE POLISH NOTATION

• Arithmetic Expressions: A + B

A + B	Infix notation
+ A B	Prefix or Polish notation
A B +	Postfix or reverse Polish notation

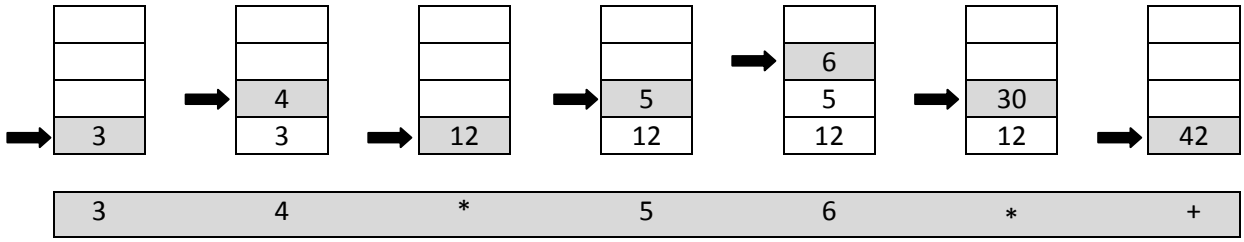
The reverse Polish notation is very suitable for stack manipulation

e.g. A * B + C * D = AB * CD * +

Evaluation of Arithmetic Expressions

- Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) = 3 4 * 5 6 * + = 42



Problems

Give Zero, one, two and three address programs for given equations:

1. A * B + C * D + E * F
2. A * B + A * (B * D + C * E)
3. [A * [B + C * (D + E)]] / [F * (G + H)]
4. (A + B * C) / (D – E * F + G * H)
5. A + B * [C * D + E * (F + G)]
6. (A + B – C) / (D * (5 * F – G))
7. (A – B + C * (D * E – F)) / (G + H * K)

A * B + C * D + E * F			
A B * C D * E F * + +			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
MUL
PUSH C
PUSH D
MUL
PUSH E
PUSH F
MUL
ADD
ADD
POP X
	LOAD A
	MUL B
	STORE T
	LOAD C
	MUL D
	ADD T
	STORE T
	LOAD E
	MUL F
	ADD T
	STORE X
		MOV R1, A
		MUL R1, B
		MOV R2, C
		MUL R2, D
		ADD R1,R2
		MOV R2, E
		MUL R2,F
		ADD R1,R2
		MOV X,R1
			MUL R1,A,B
			MUL R2,C,D
			ADD R1,R1,R2
			MUL R2,E,F
			ADD X,R1,R2

A * B + A * (B * D + C * E)			
A B * A B D * C E * + * +			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
MUL
PUSH A
PUSH B
PUSH D
MUL
PUSH C
PUSH E
MUL
ADD
MUL
ADD
POP X
	LOAD C
	MUL E
	STORE T
	LOAD B
	MUL D
	ADD T
	MUL A
	STORE T
	LOAD A
	MUL B
	ADD T
	STORE X
		MOV R1,C
		MUL R1,E
		MOV R2,B
		MUL R2,D
		ADD R1,R2
		MUL R1,A
		MOV R2,A
		MUL R2,B
		ADD R1,R2
		MOV X,R1
			MUL R1,C,E
			MUL R2,B,D
			ADD R1,R1,R2
			MUL R1,R1,A
			MUL R2,A,B
			ADD X,R1,R2

[A * [B + C * (D + E)]] / [F * (G + H)]			
A B C D E + * + * F G H + * /			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
PUSH C
PUSH D
PUSH E
ADD
MUL
ADD
MUL
PUSH F
PUSH G
PUSH H
ADD
MUL
DIV
POP X
	LOAD H
	ADD G
	MUL F
	STORE T
	LOAD E
	ADD D
	MUL C
	ADD B
	MUL A
	DIV T
	STORE X
		MOV R1,E
		ADD R1,D
		MUL R1,C
		ADD R1,B
		MUL R1,A
		MOV R2,H
		ADD R2,G
		MUL R2,F
		DIV R1,R2
		MOV X,R1
			ADD R1,D,E
			MUL R1,R1,C
			ADD R1,R1,B
			MUL R1,R1,A
			ADD R2,G,H
			MUL R2,R2,F
			DIV X,R1,R2

(A + B * C) / (D – E * F + G * H)			
A B C * + D E F * – G H * + /			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
PUSH C
MUL
ADD
PUSH D
PUSH E
PUSH F
MUL
PUSH G
PUSH H
MUL
ADD
DIV
POP X
	LOAD E
	MUL F
	STORE T
	LOAD D
	SUB T
	STORE T
	LOAD G
	MUL H
	ADD T
	STORE T
	LOAD B
	MUL C
	ADD A
	DIV T
	STORE X
		MOV R2,E
		MUL R2,F
		MOV R1,D
		SUB R1,R2
		MOV R2, G
		MUL R2,H
		ADD R1,R2
		LOAD R2,B
		MUL R2,C
		ADD R2,A
		DIV R2,R1
		MOV R2,X
			MUL R1,E,F
			SUB R2, D,R1
			MUL R1,G,H
			ADD R2,R1,R2
			MUL R1,B,C
			ADD R1,R1,A
			DIV X,R1,R2

A + B * [C * D + E * (F + G)]			
A B C D * E F G + * + * +			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
PUSH C
PUSH D
MUL
PUSH E
PUSH F
PUSH G
ADD
MUL
ADD
MUL
ADD
POP X
	LOAD G
	ADD F
	MUL E
	STORE T
	LOAD C
	MUL D
	ADD T
	MUL B
	ADD A
	STORE X
		MOV R1,F
		ADD R1,G
		MUL R1,E
		MOV R2,C
		MUL R2,D
		ADD R1,R2
		MUL R1,B
		ADD R1,A
		MOV X,R1
			ADD R1,F,G
			MUL R1,R1,E
			MUL R2,C,D
			ADD R1,R1,R2
			MUL R1,R1,B
			ADD X,R1,A

(A + B – C) / (D * (5 * F – G))			
A B + C – D 5 F * G – * /			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
ADD
PUSH C
SUB
PUSH D
PUSH 5
PUSH F
MUL
PUSH G
SUB
MUL
DIV
POP X
	LOAD 5
	MUL F
	SUB G
	MUL D
	STORE T
	LOAD A
	ADD B
	SUB C
	DIV T
	STORE X
		MOV R1,A
		ADD R1,B
		SUB R1,C
		MOV R2,F
		MUL R2,5
		SUB R2,G
		MUL R2,D
		DIV R1,R2
		MOV X,R2
			MUL R2,5,F
			SUB R1,R1,G
			MUL R2,R2,D
			ADD R1,A,B
			SUB R1,R1,C
			DIV X,R1,R2

(A – B + C * (D * E – F)) / (G + H * K)			
A B – C D E * F – * + G H K * + /			
0 Address	1 Address	2 Address	3 Address (DEST, SRC, SRC)

PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MUL
PUSH F
SUB
MUL
ADD
PUSH G
PUSH H
PUSH K
MUL
ADD
DIV
POP X
	LOAD H
	MUL K
	ADD G
	STORE T
	LOAD D
	MUL E
	SUB F
	MUL C
	ADD A
	SUB B
	DIV T
	STORE X
		MOV R2,H
		MUL R2,K
		ADD R2,G
		MOV R1,D
		MUL R1,E
		SUB R1,F
		MUL R1,C
		ADD R1,A
		SUB R1,B
		DIV R1,R2
		MOV X,R1
			MUL R2,H,K
			ADD R2,R2,G
			MUL R1,D,E
			SUB R1,R1,F
			MUL R1,R1,C
			SUB R3,A,B
			ADD R1,R1,R3
			DIV X,R1,R2