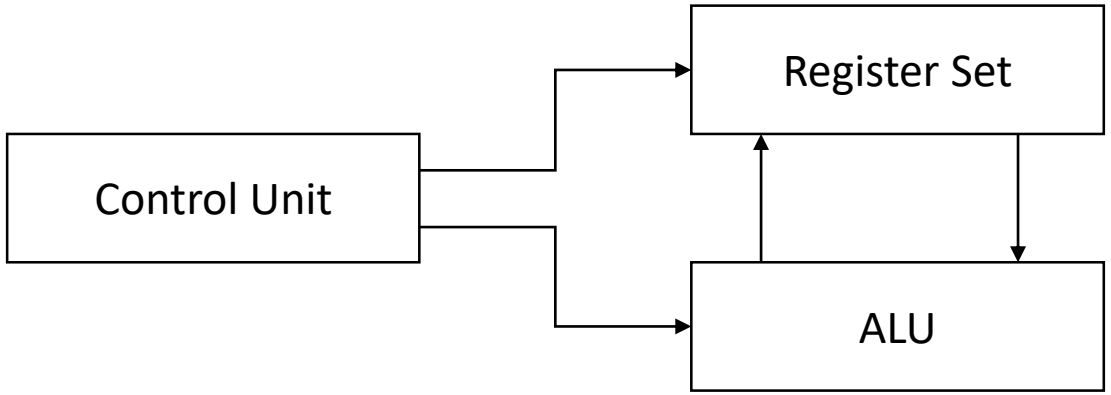
Unit 4: Central Processing Unit

Reference : Chapter 8 Computer System Architecture by Morris Mano

Central Processing Unit (CPU)

- It is a part of computer that performs the bulk of data-processing operations.

- CPU is made up of 3 major part:



- Register Set:
  - Stores intermediate data used during the execution of the instructions.
- Arithmetic logic unit (ALU):
  - Performs the required micro operations for executing the instructions.
- Control unit:
  - Supervise the transfer of information among the registers and instructs the ALU which operation to perform.

The computer structure and Behavior

- As seen by programmer that uses machine language instructions includes
  - Instruction formats
  - Addressing modes
  - The instruction set
  - General organization of the CPU registers

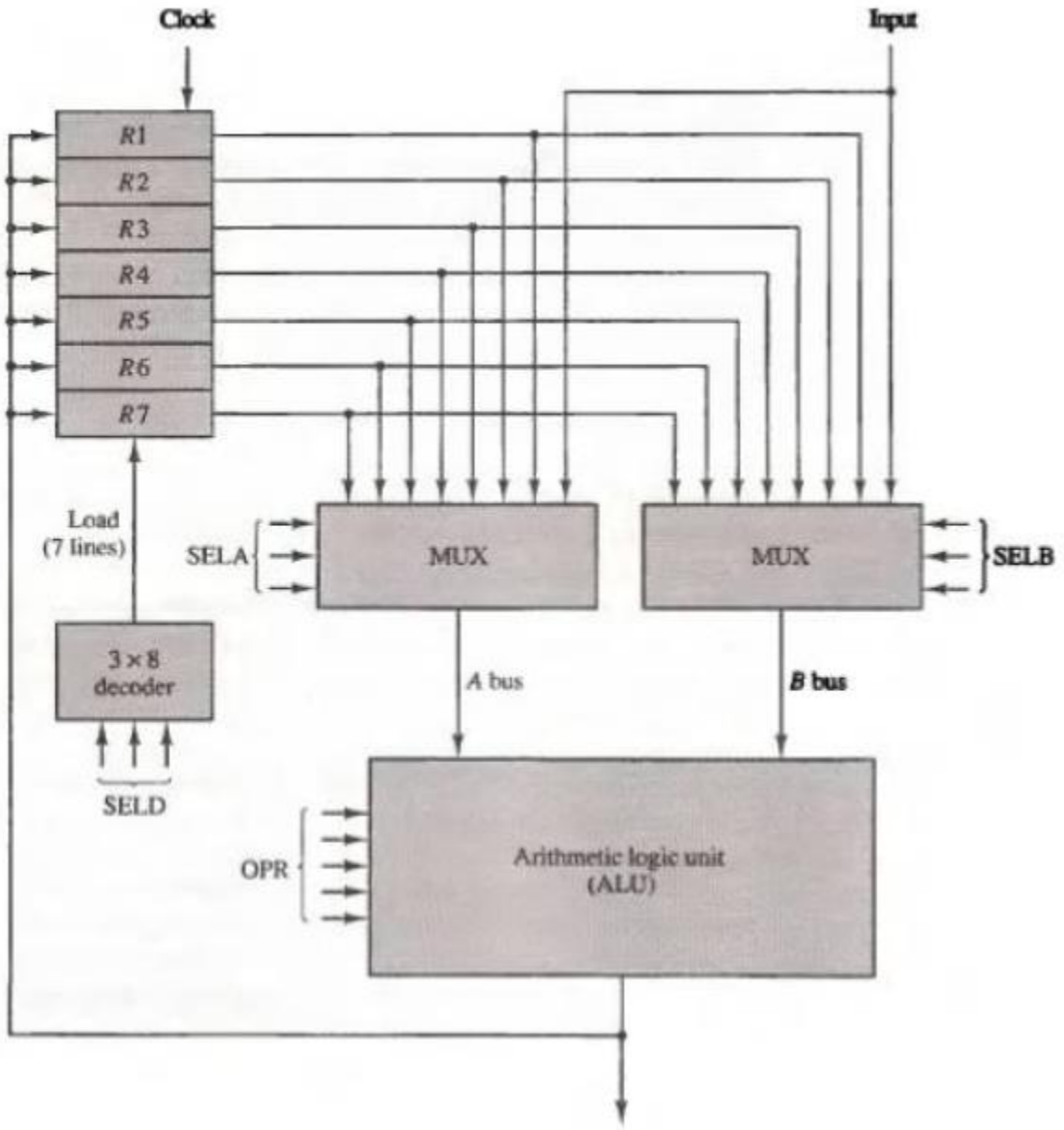
- The user who programs in machine or assembly language must be aware of the register set, the memory structure, the type of data supported by the instructions and the function that each instruction perform.

General Register Organization

- Why Register?
  - Memory access is time consuming.

- Register is used to store:
  - Pointers, Counters, return addresses, temporary results and partial product during multiplication.

- Large number of registers in CPU are connected through a common bus system.

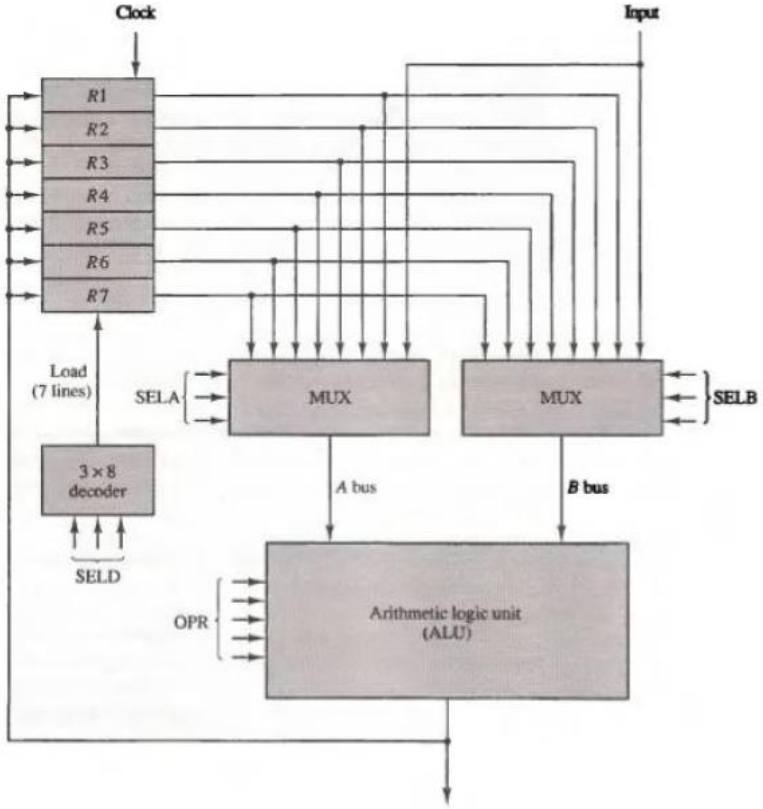


Register Set with common ALU

Operation (R1← R2 + R3) and information flow

- Control must provide binary selection to variables to the following selector inputs.

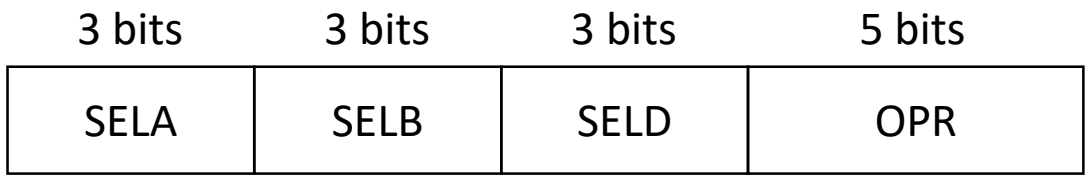
1. MUX A selector (SELA):
   - To place the content of R2 into bus A.
2. Mux B selector (SELB):
   - To place the content of R3 into bus B.
3. ALU operation selector (OPR):
   - To provide arithmetic operation A + B.
4. Decoder Destination selector (SELD):
   - To transfer the content of the output bus into R1.



- The four control selection variables are generated in the control unit and must be available at the beginning of the clock cycle.

Control word

- Selection inputs in the unit and their combined value specifies a control word.



- SELA: select a source register for the A input.
- SELB: select a source register for the B input.
- SELD: select a destination register using the decoder and its seven load outputs.
- OPR: select the one of the operation in the ALU.

Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Example

1. Give Binary control word for R1 ← R2 - R3

	3 bits	3 bits	3 bits	5 bits
Field	SELA	SELB	SELD	OPR
Symbol	R2	R3	R1	SUB
Control word	010	011	001	00101

Example

1. R4 ← R4 ˅ R5 :

   Control word: 100 101 100 01010

2. Output ← R2 :

   Control word: 010 000 000 00000

3. R5 ← 0: (Reset)

   R5 ⊕ R5 = 0

   Control word: 101 101 101 01100

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

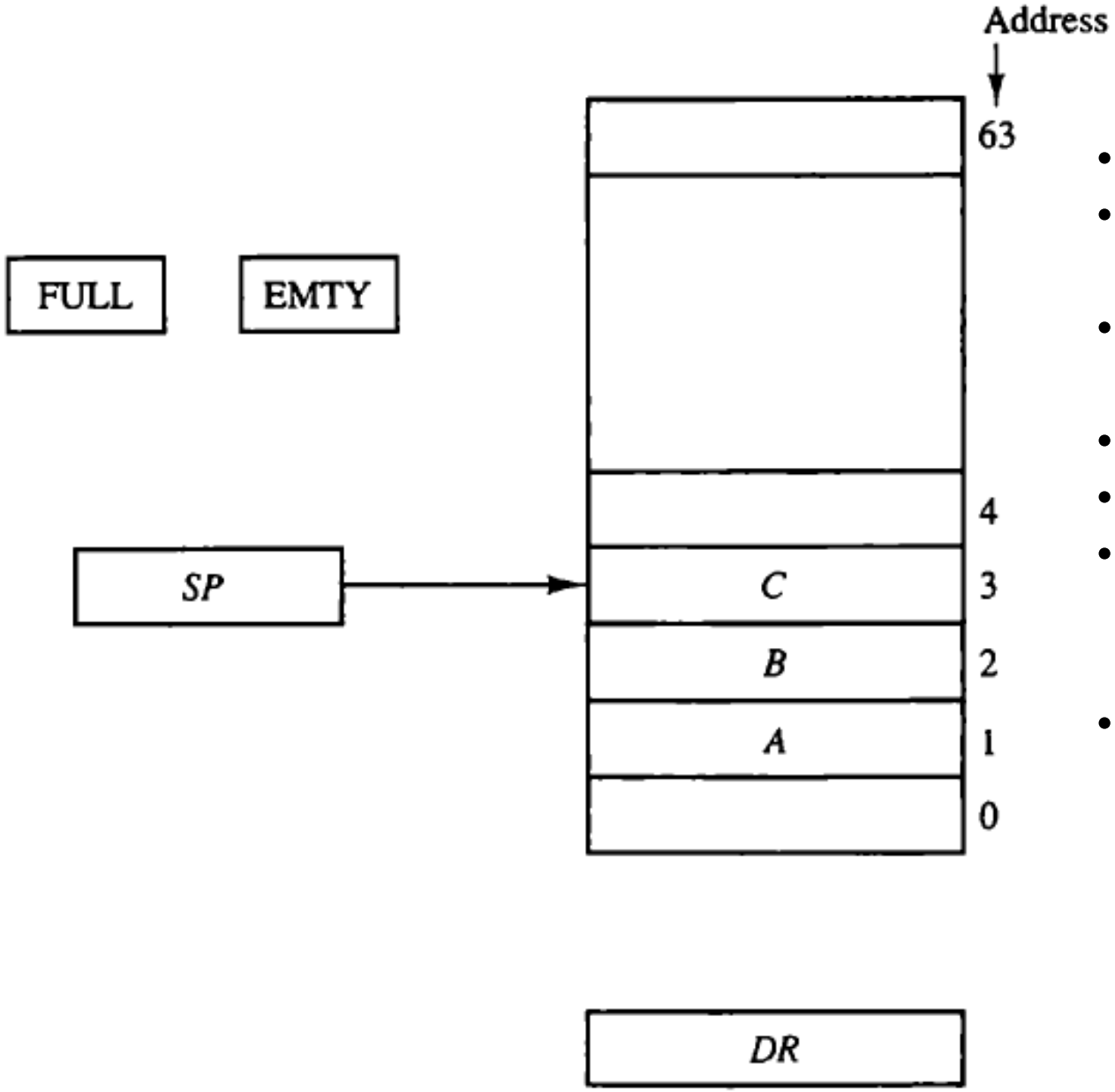
- Control Memory: A memory unit that stores the control words.

- Micro programmed control: By reading consecutive control words from memory, it is possible to initiate the desired sequence of micro operation for the CPU. This type of control is referred to as micro programmed control.

Stack Organization

- LIFO (Last In First Out)
- Stack Pointer (SP):
  - The register that holds the address for the stack.
  - It points at the top item in stack.
- Two Operation: PUSH and POP
- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

64- word register stack organization



- SP contains 6 bits because 64 words.
- When 63(111111) incremented by 1. The result is 0.
- When 000000 decremented by 1 the result is 111111.
- FULL=1 when stack is full.
- EMTY=1 when stack is empty.
- DR is the data register that holds the binary data to be written into or read out of the stack.
- Initially, SP=0 , EMTY=1 and FULL=0

Insert a new item

- If the stack is not full (if FULL=0), a new item can be inserted with PUSH operation.

SP← SP+1
M[SP]← DR
If(SP=0) then (FULL→ 1)
EMTY ← 0

- The first item stored in the stack is at address 1.
- The last item stored in the stack is at address 0.

Delete an item

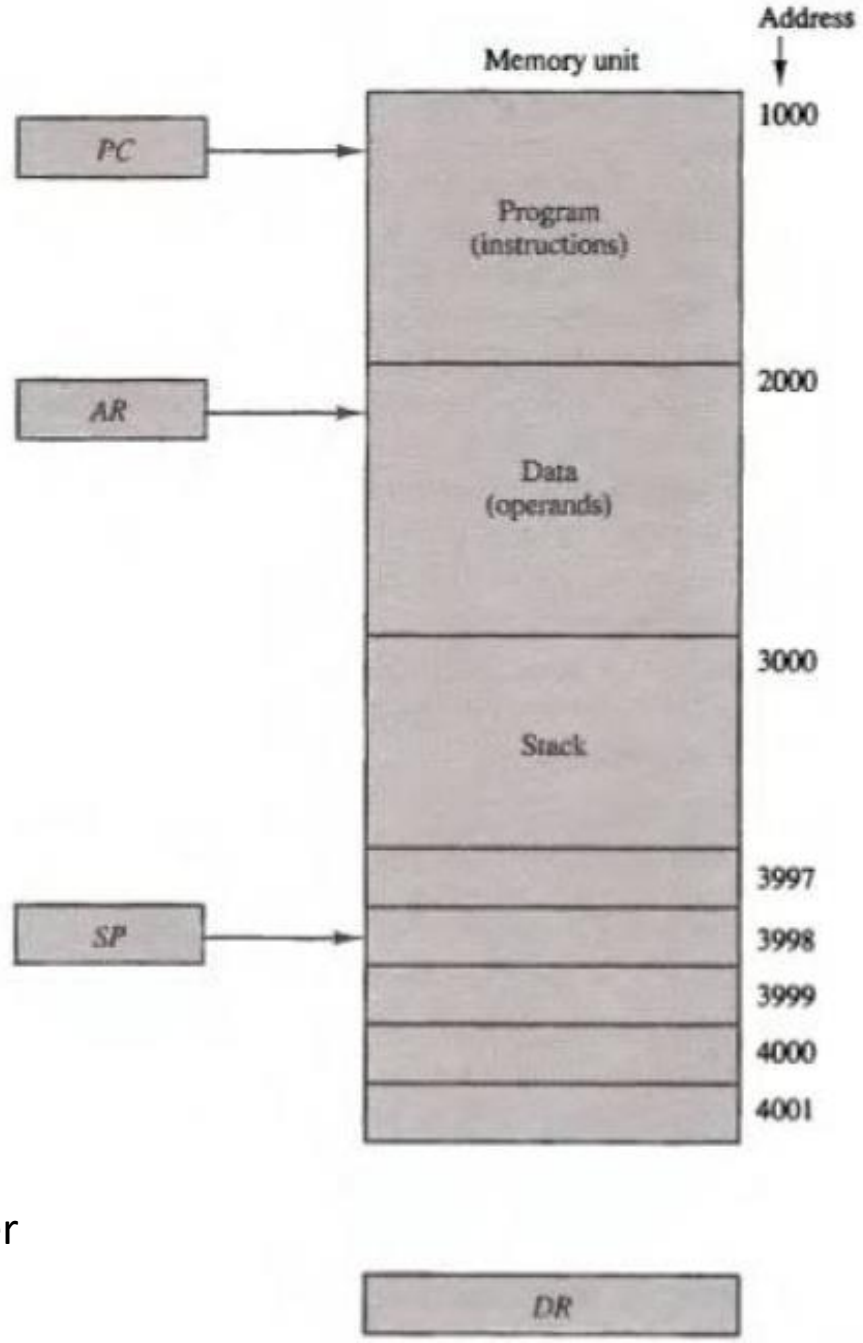
- An item is deleted from the stack if the stack is not empty (if EMTY=0).

DR← M[SP]
SP← SP – 1
If (SP=0) then (EMTY ← 1)
FULL← 0

- An erroneous operation will result if the stack is pushed when FULL=1 or popped when EMTY=1.

Memory Stack

- Computer memory partitioned into three segment: program, data and stack.
- Program Counter (PC):
  - Points at the address of the next instruction in the program.
  - Used during the fetch phase to read an instruction.
- Address Register (AR):
  - Points an array of data.
  - Used during the execute phase to read an operand.
- Stack pointer (SP):
  - Points at the top of the stack.
  - Used to push or pop items into or from the stack.



The three registers are connected to a common address bus, and either can provide as address for memory.

Continue…

- PUSH:

  SP← SP - 1

  M[SP]← DR

- POP:

  DR← M[SP]

  SP← SP + 1

- The stack limits can be checked by using two processor register. One to hold upper limit (3000 in this case) and other to hold lower limit (4001 in this case).
- CPU can refer to memory without having to specify an address, since the address is always available and automatically updated in the SP.

Instruction Formats

- Control Unit within the CPU interpret each instruction code and provide necessary control functions needed to process the instruction.

- Common fields found in instruction formats:
  1. Operation code: that specifies the operation to be performed
  2. Address: that designates a memory address or a processor register.
  3. Mode: that specifies the way the operand or the effective address is determined.

- The number of address fields in the instruction format of a computer depends on the internal organization.

CPU Organization

Three types of CPU organization:

1. Single accumulator organization
2. General register organization
3. Stack organization

Single Accumulator Organization

- All operation are performed with an implied accumulator register.
- Instruction format in this type of computer uses one address field.
- ADD X to denote AC ← AC + M[X]
- AC is the accumulator register.
- M[X] symbolizes the memory word located at address X.
- One Address type instruction.

General Register Organization

- Instruction format in this type of computer uses three or two address fields.
- ADD R1, R2, R3 to denote R1 ← R2 + R3
- ADD R1, R2 to denote R1← R2 + R2
- Each address field may specify a processor register or a memory word.
- Three and two address type instruction.

Stack Organization

- Have PUSH and POP instructions which require an address field.
- PUSH X will push the word at address X to top of the stack.
- Operation type instructions do not need an address field.
- ADD will pop two numbers from stack , Add them, and push the sum into the stack.
- Zero Address type instruction.

Example: (Zero, One, Two, Three type address Instructions)

- Evaluate the arithmetic statement X = (A + B) * (C +D) using Zero, One, Two, Three type address Instructions.
- Arithmetic Operation: ADD, SUB, MUL.
- MOV transfer type operation.
- LOAD and STORE for transfer to and from memory and AC register.
- Operands are in memory address A,B,C and D.
- Result must be stored in add X.

Three Address Instruction

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

- Advantage of this format is that it result in short programs.
- Disadvantage is that the binary coded instructions require too many bits to specify three address.

Two Address Instruction

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

The first symbol listed in an instruction is assumed to be both a source and the destination.

One Address Instruction

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

- T is address of temporary memory location.
- All operation are done between the AC register and memory operand.

Zero Address Instruction

- Does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions need an address field to specify the operand that communicates with the stack.
- To evaluate expression in this, it is necessary to convert the expression into reverse polish notation.

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

TOS = Top of the Stack

RISC Instructions

- Instruction set in RISC processor is restricted to the use of LOAD and STORE instructions when communicating between memory and CPU.
- All other instructions are executed within the registers of the CPU without referring to memory.

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

- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computer use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
  1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data and program relocation.
  2. To reduce number of bits in the addressing field of the instruction.
- Give flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.

Instruction Cycle:

- Divided into three major phases:
  1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Execute the instruction.

1. Program Counter (PC) register holds the address of the next instruction to be executed and is incremented each time an instruction is fetched from memory.
2. The decoding is done in next step determines the operation to be performed, the addressing mode of the instruction and the location of the operands.
3. The computer then executes the instruction and return to step 1 to fetch the next instruction in the sequence.

Instruction Format with mode field:

- The operation code specifies the operation to be performed.
- The mode field is used to locate the operands needed for the operation.
- There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register.

Addressing Mode:

Implied Mode:

- In this mode the operands are specified implicitly in the definition of the instruction.
- Ex: Complement accumulator.
- No address field is required.
- All register reference instruction that use an accumulator are implied mode instructions.
- Zero-address instructions in a stack organized computer are implied mode instruction since the operands are implied to be on the top of the stack.

Continue…

Immediate Mode:

- Operand is specified in the instruction itself.
- We have an operand field rather than an address field.
- Useful for initializing registers to a constant value.

Register Mode:

- Operands are in a processor register.
- Address fields of an instruction specifies a processor register.
- The particular register is selected from a register field(= address field) in the instruction.
- A k-bit field can specify any one of 2^K registers.

Continue…

Register Indirect Mode:

- Instruction specifies a register in the CPU whose contents give the address of the operand in memory.

- Advantage: address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Continue…

Autoincrement or Autodecrement Mode:

- Similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.

- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

Continue…

Direct Address Mode:

- Effective address is equal to the address part of the instruction.

- Effective address: is defined to be the memory address obtained from the computation dictated by the given addressing mode.

Indirect Address Mode:

- Address field of the instruction gives the address where the effective address is stored in memory.

Continue…

Relative Address Mode:

- Content of the PC is added to the address part of the instruction in order to obtain the effective address.
- Effective Address = PC + Address
- The position in memory is relative to the address of the next instruction.

- Ex: PC= 825, Address part of the instruction= 24

- Effective Address= 826 + 24 = 850

Continue…

Indexed Addressing Mode:

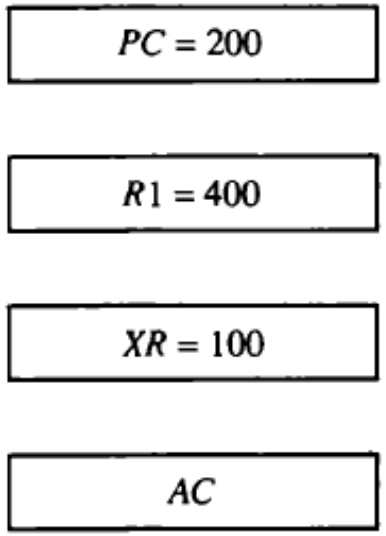
- Content of the Indexed register is added to the address part of the instruction in order to obtain the effective address.
- Effective Address = Indexed register + Address
- The address field defines the beginning address of a data array in memory.
- If it does not include address field in its format, the instruction converts to the register indirect mode.

Continue…

Base Register Addressing Mode:

- Content of the base register is added to the address part of the instruction in order to obtain the effective address.
- Effective Address = Base register + Address
- Used to facilitate the relocation of programs in memory.

Example:



Instruction: Load 500

Address	Memory	
200	Load to *AC*	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

For each possible mode calculate Effective address and the operand that must be loaded into AC.

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

Mnemonic for Addressing Modes

- The mnemonic for load immediate becomes LDI.
- In other assembly level language the immediate mode is recognized from sign # placed before the operand.
- Below is Load instruction when used with different addressing modes.

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

Data Transfer and Manipulation

Computer instruction can be classified into three categories:

1. Data transfer instructions:

   Cause transfer of data from one location to another without changing the binary information content.

2. Data Manipulation instructions:

   Perform arithmetic. Logic and shift operations.

3. Program control instructions:

   Provide decision making capabilities and change the path taken by the program when executed in the computer.

Data Transfer Instructions

- Move data from one place in the computer to another without changing the data content.

- The most common transfer are between:
  - Memory and processor registers.
  - Processor registers and Input or Output.
  - Processor registers themselves.

Continue…

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

- Transfer from Memory to Processor register, usually an accumulator.
- Transfer from Processor register into Memory.
- Transfer from one register to another, registers and memory or between two memory words.
- Swaps information between two registers or a register and a memory word.
- Transfer data among processor registers and input or output terminal.
- Transfer data between processor registers and a memory stack.

Data Manipulation Instructions

- Perform operation on data and provide the computational capabilities for the computer.

- It is divided into three basic types:
  1. Arithmetic Instructions
  2. Logical and bit manipulation instructions
  3. Shift instructions

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit Manipulation Instructions

- Logical Instructions perform binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary coded information.
- It is possible to change bit value, to clear a group of bits or to insert new bit values into operands stored in registers or memory words.

Continue…

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Bit manipulation

- There are three bit manipulation operations possible:
  1. Can be cleared to 0
  2. Can be set to 1
  3. Can be complemented

- Clear selected bits: ANDed with a 0 produces a 0.
  - The AND instruction is also called mask because it masks or inserts 0's in a selected portion of an operand.
- Set selected bits: ORed with a 1 produces a 1.
  - Selectively set bits of an operand by ORing it with another operand with 1's in the bit positions that must be set to 1.
- Complement selected bits: XORed with 1 complements a bit.

Shift Instructions

- There are three type of shift possible with each left and right shift:
  1. logical
  2. Arithmetic
  3. Rotate
  - Arithmetic left shift is same as logical left shift

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Possible instruction code format of a shift instruction

Program Control

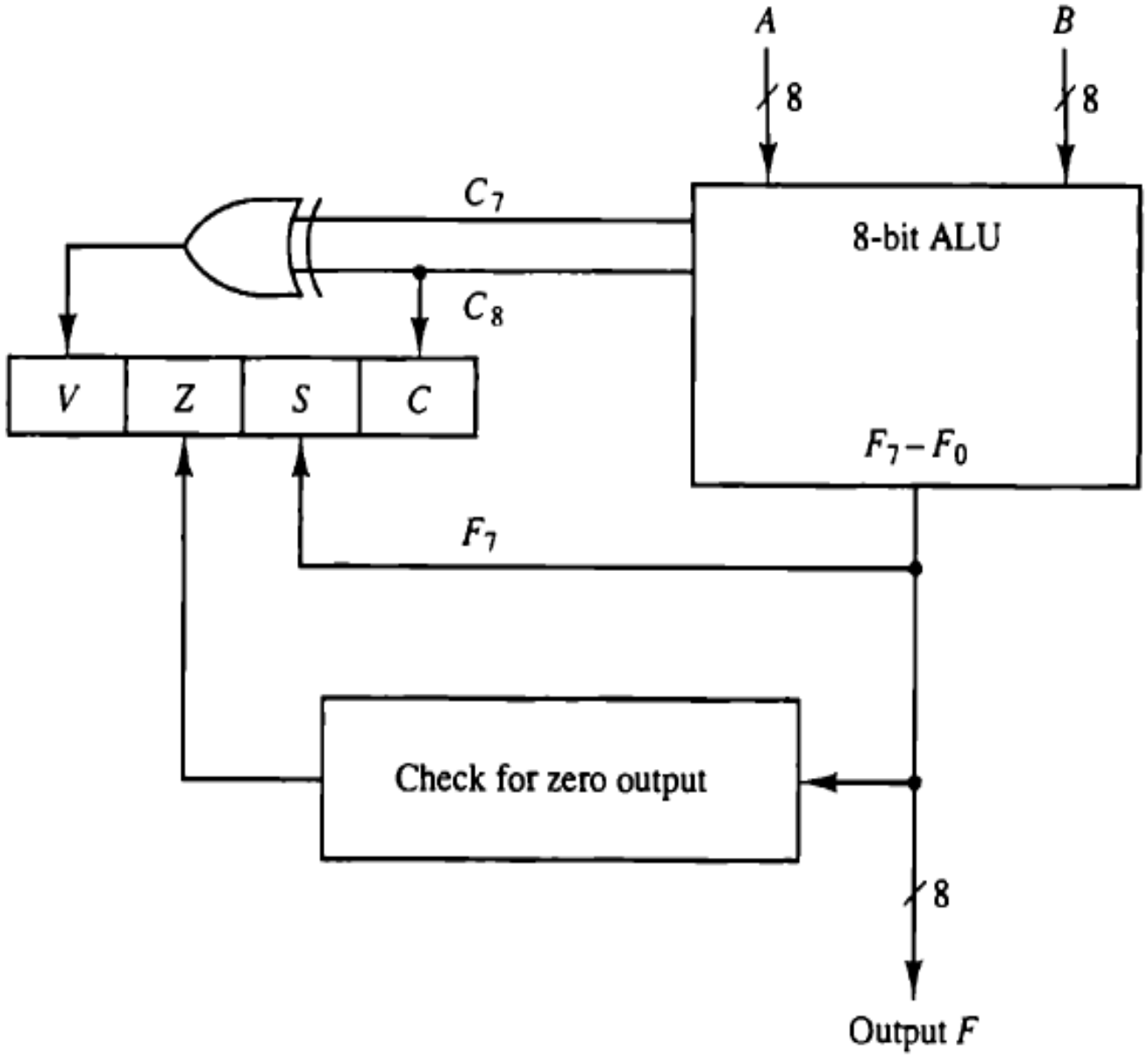
- A program control type instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Continue…

- Branch and Jump instructions are used interchangeably to mean the same thing, but some times they are used to denote different addressing modes.
- Branch and Jump instructions may be conditional(if positive or if Zero) or unconditional.
- Skip is a zero-address instruction. It skip the next instruction if the condition is met.
- Compare instruction performs a subtraction between two operands, but the result is not retained.
- Test instruction performs the logical AND of two operands without retaining the result or changing the operands.
- However, In compare and Test certain status bits are set.

Status Bits/Condition code/Flag Bits:



Carry Bit(C)

Sign Bit(S)

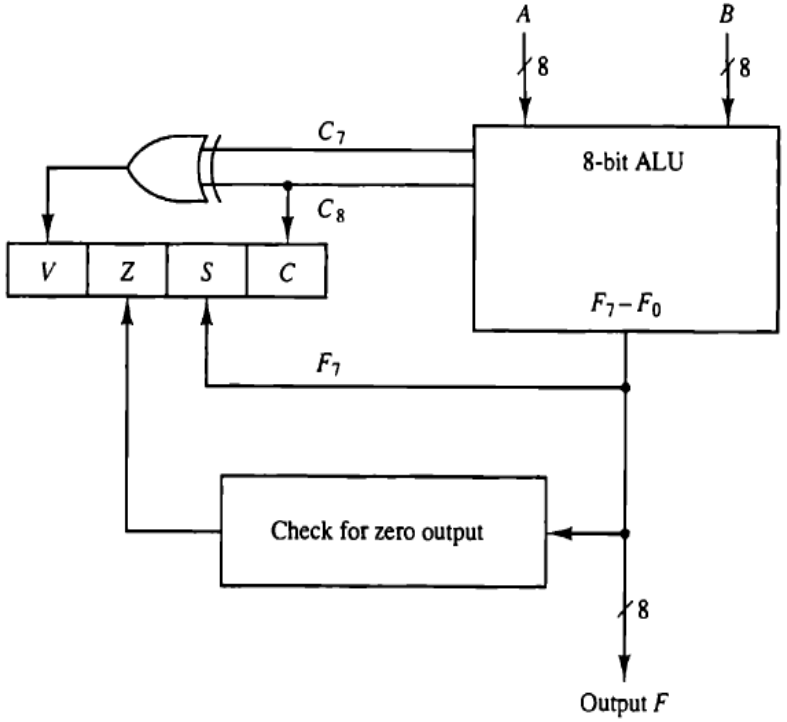
A Zero Indication(Z)

An overflow condition(V)

[V Z S C] 4-Bit Status Register

Continue…

- Bit C (Carry):
  - =1 if the end carry C_8 is 1.
  - =0 if the carry is 0.
- Bit S (Sign):
  - =1 if the highest order bit F_7 is 1.
  - =0 if the highest order bit F_7 is 0.
- Bit Z (Zero):
  - =1 if the output of the ALU is 0.
  - =0 if the output of the ALU is not 0.
- Bit V (Overflow):
  - =1 if the Exclusive-OR of the last two carries is 1. (8-bit ALU, V=1 if output is > +127 or < -128)
  - =0 otherwise
  - This is condition when numbers are in 2's complement form.



Continue…

- If V is set after the addition of two sign numbers, it indicates an overflow condition.
- If Z is set after an exclusive-OR operation, it indicates that A=B.
- A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.
- Ex: A=101x1100 AND B=00010000
- Produces a result 000x0000.
- If x=0 the Z status bit is set to 1, but if x=1 the Z status bit is cleared to 0.

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	$A \neq B$

Example

A= 11110000 B=00010100 , Perform A - B

V	Z	C	S
0	0	1	1

$$\begin{array}{r} 11110000 \\ +\ 11101100 \\ \hline 11011100 \end{array}$$

2's Complement of B

Unsigned Number

- A=240, B=20
- 240-20 = 220
- A > B as C = 1 and A ≠ B as Z = 0
- Branch After this comparison are:
  BHI, BHE, BNE

Signed Number

- A= -16, B= +20
- (-16) – (+20) = (-36)
- A < B and A ≠ B as S=1,V=0,Z=0
- Branch After this comparison are:
  BLT, BLE, BNE

The subtraction is same for these two representation

Subroutine Call and Return

- A subroutine is a self-contained sequence of instructions that performs a given computational task.
- Can be called many times.
- The instruction that transfers program control to a subroutine is known by:
  - Call subroutine, Jump to subroutine, Branch to subroutine, Branch and save address

Subroutine Call and Return

- A call subroutine instruction consist of an operation code together with an address that specifies the beginning of the subroutine.
- The Instruction is executed by performing two operation:
  1. The address of the next instruction available in PC(the return address) is stores in a temporary location.
  2. Control is transferred to the beginning of the subroutine.

Continue…

- The last instruction for every subroutine, commonly called return from subroutine, transfer the return address from the temporary location into the program counter.

- Different temporary location for storing the return address:
  - First memory location of the subroutine
  - Fixed location in memory
  - In a processor register
  - In a memory stack (Efficient way)

Advantage of Stack:

- When succession of subroutines is called, the sequential return addresses can be pushed into the stack.
- The return from subroutine instruction causes the stack to pop and the contents of the top of the stack are transferred to program counter.
- The return is always to the program that last called a subroutine.
- Solve the problem of recursive subroutine.

Implementation

- Subroutine Call:

  SP ← SP – 1 (Decrement stack pointer)

  M[SP] ← PC (Push content of PC onto the stack)

  PC ← Effective address (Transfer control to the subroutine)

- Return from Subroutine:

  PC ← M[SP] (POP stack and transfer to PC)

  SP ← SP + 1 (Increment stack pointer)

Program Interrupt

- Used to handle variety of problems that arise out of normal program sequence.
- It refers to the transfer of program control from a current running program to another service program as a result of an external or internal generated request.

Difference between Interrupt and Subroutine:

1. The interrupt usually initiated by an internal or external signal rather than from the execution of an instruction(except for software interrupt).
2. The address of the interrupt service program is determined by the hardware rather than from the address field of the instruction.
3. The interrupt procedure usually stores all the information necessary to define the state of the CPU rather than just storing the PC.

Continue…

- CPU must return to exactly the same state that it was when the interrupt occurred.
- The state of the CPU at the end of the execute cycle is determined from:
  - Content of the PC.
  - Content of all processor registers.
  - Content of certain status conditions.
- The collection of all status bit conditions in the CPU is called Program status word(PSW).
- PSW stored in separate hardware register.
- It includes status bit from the last ALU operation and it specifies the interrupts that are allowed to occur and whether the CPU is operating in a user mode or supervisor mode.

Continue…

- CPU does not respond to an interrupt until the end of an instruction execution.
- Just before going to the next fetch phase, control checks for any interrupt signals.
- If an interrupt is pending, control goes to a hardware interrupt cycle.
- During this cycle, content of PC and PSW are pushed onto the stack.
- The branch address for the particular interrupt is then transferred to PC and new PSW is loaded into the status register. Now service program can be executed.
- The last instruction in the service program is a return from interrupt instruction.
- The stack is popped to retrieve old PSW and return address.
- Thus the CPU state is restored and the original program can continue executing.

Types of Interrupt

1. External Interrupt: Come from I/O devices, from a timing device, from circuit monitoring the power supply, from any other external source.

   Example: I/O device requesting transfer of data, Elapsed time of an event, Power failure.

Types of Interrupt

2.Internal interrupt: Arise from illegal or erroneous use of an instruction or data. It is also called traps.

Example: Register overflow, attempt to divide by zero, an invalid operation code, stack overflow and protection violation.

Types of Interrupt

3. Software Interrupt: initiated by executing an instruction.
   - Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call.
   - Used by programmer to initiate an interrupt procedure at any desired point in the program.
   - Supervisor call instruction.(Switch from user mode to supervisor mode)

Difference between Internal and External Interrupt

1. Internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.
2. Internal interrupt are synchronous with the program while external interrupts are asynchronous.
3. If the program is executed the internal interrupts will occur in the same place each time. External interrupts depends on external condition.

CISC and RISC

- A computer with large number of instructions is classified as Complex Instruction Set Computer(CISC).
  - Goal: Provide single machine level instruction for each statement that is written in high level language.
  - Ex: VAX computer, IBM 370.

- Computers should use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. This type of computer is classified as a Reduced Instruction Set Computer(RISC).
  - Goal: Attempt to reduce execution time by simplifying the instruction set.

CISC Characteristics:

1. A large number of instructions – typically from 100 to 250 instructions.
2. Some instructions that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes- typically from 5 to 20.
4. Variable length instruction formats.
5. Instructions that manipulate operands in memory.

RISC Characteristics:

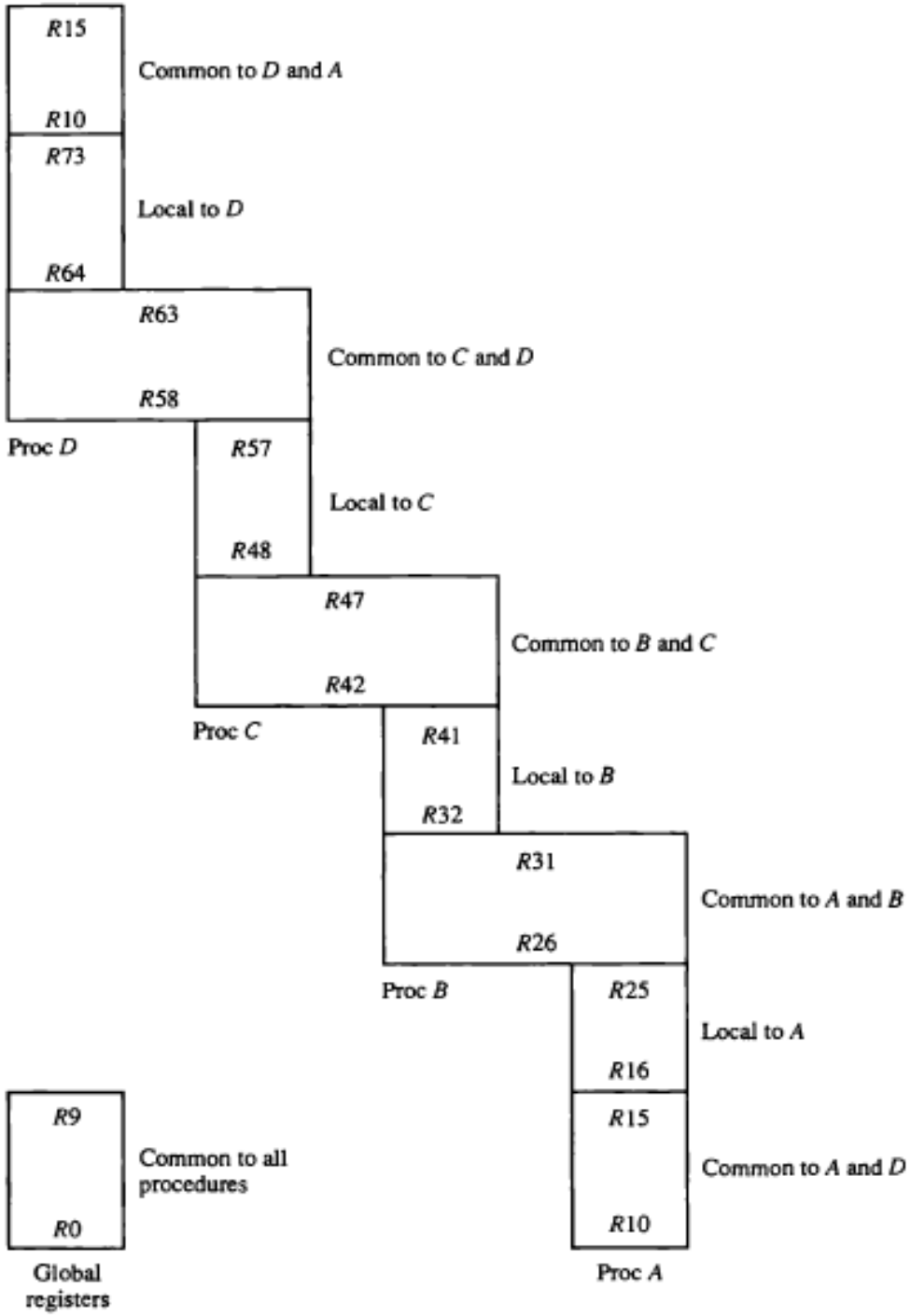
1. Relatively few instructions.
2. Relatively few addressing modes.
3. Memory access limited to load and store instructions.
4. All operations done within the registers of the CPU.
5. Fixed length, easy decode instruction format.
6. Single cycle instruction execution.(By overlapping fetch, decode & execute phase of two or three instructions)
7. Hardwired rather than micro programmed control.

Continue…

8. A relatively large number of registers in the processor unit.
9. Use of overlapped register windows to speed up procedure call and return.
10. Efficient instruction pipeline.
11. Compiler support (for efficient translation of high-level language programs into machine language programs.

Overlapped Register Windows

- Provide the passing of parameters and avoid the need for saving and restoring register values.
- Each procedure call results in the allocation of new window consisting of a set of registers from the register file.
- Each procedure call activates a new register window by incrementing pointer, while the return statement decrements the pointer and causes the activation of the previous window.
- Windows for adjacent procedures have overlapping registers that are shared to provide the passing of parameters and results.
- Only one register window is activated at any given time with a pointer indicating the active window.



- # of global registers=G
- # of local registers in each window=L
- # of registers common to two windows=C
- # of windows=W
- Window size

  = L + 2C + G

  = 10 + 12 + 10 = 32
- Register needed in the processor

  =(L + C)W + G

  =(10 + 6) * 4 + 10 = 74