IAA6007: Computer Architecture
Ch.8. Central Processing Unit

Wooil Kim
Dept. of Computer Science & Engineering
Incheon National University

2019 Fall

Outline

- 8.1 Introduction
- 8.2 General register organization
- 8.3 Stack organization
- 8.4 Instruction formats
- 8.5 Addressing modes
- 8.6 Data transfer and manipulation
- 8.7 Program control
- 8.8 Reduced Instruction Set Computer (RISC)

8.1 Introduction

- Computer architecture seen by programmer
  - Instruction formats
  - Addressing modes
  - Instruction set
  - General organization of the CPU registers

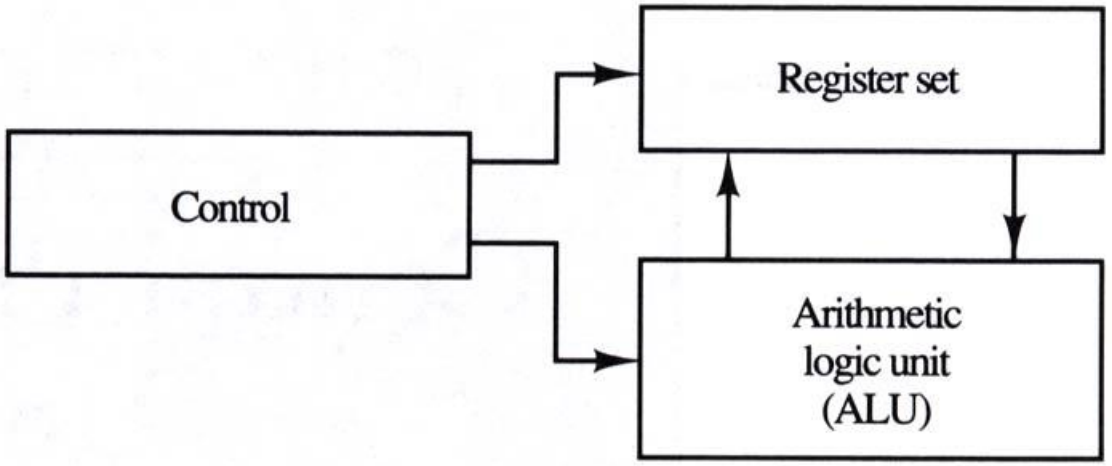


Figure 8-1 Major components of CPU.

8.1 Introduction

- Chapter 5

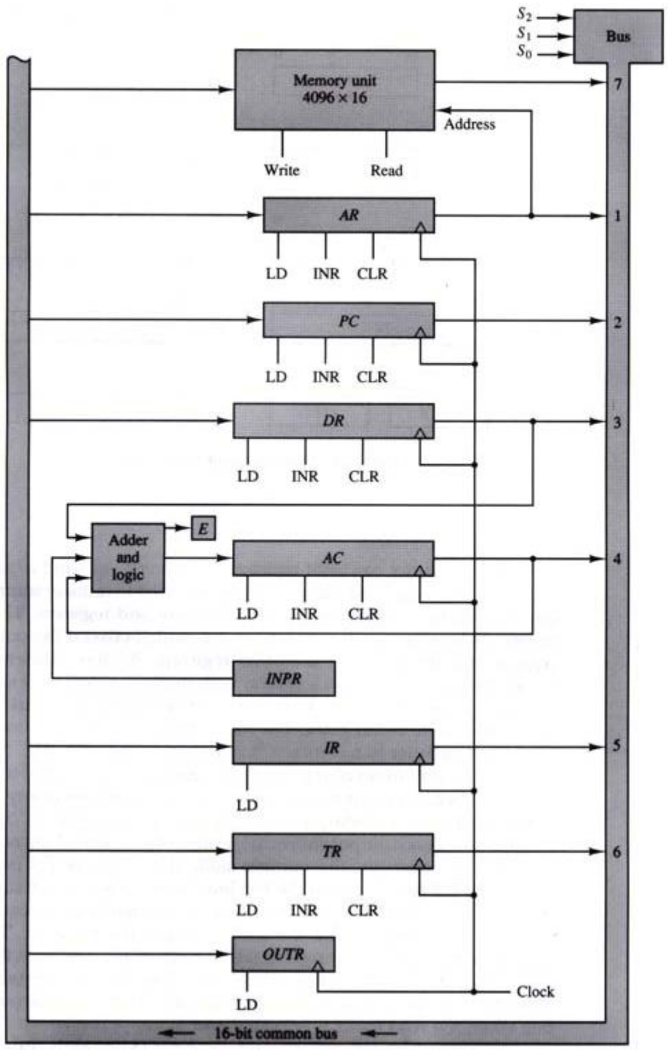


Figure 5-4 Basic computer registers connected to a common bus.

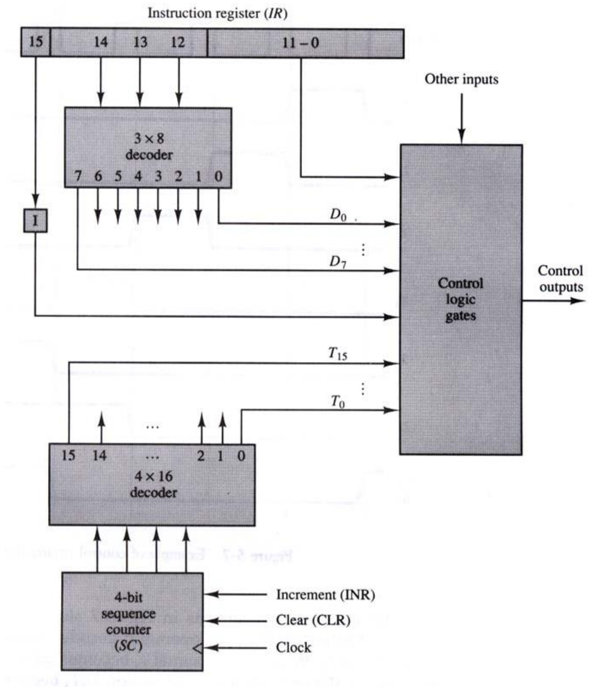
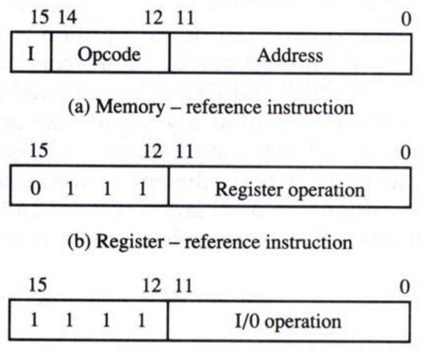


Figure 5-6 Control unit of basic computer.



(a) Memory – reference instruction

(b) Register – reference instruction

(c) Input – output instruction

8.1 Introduction

- Chapter 7

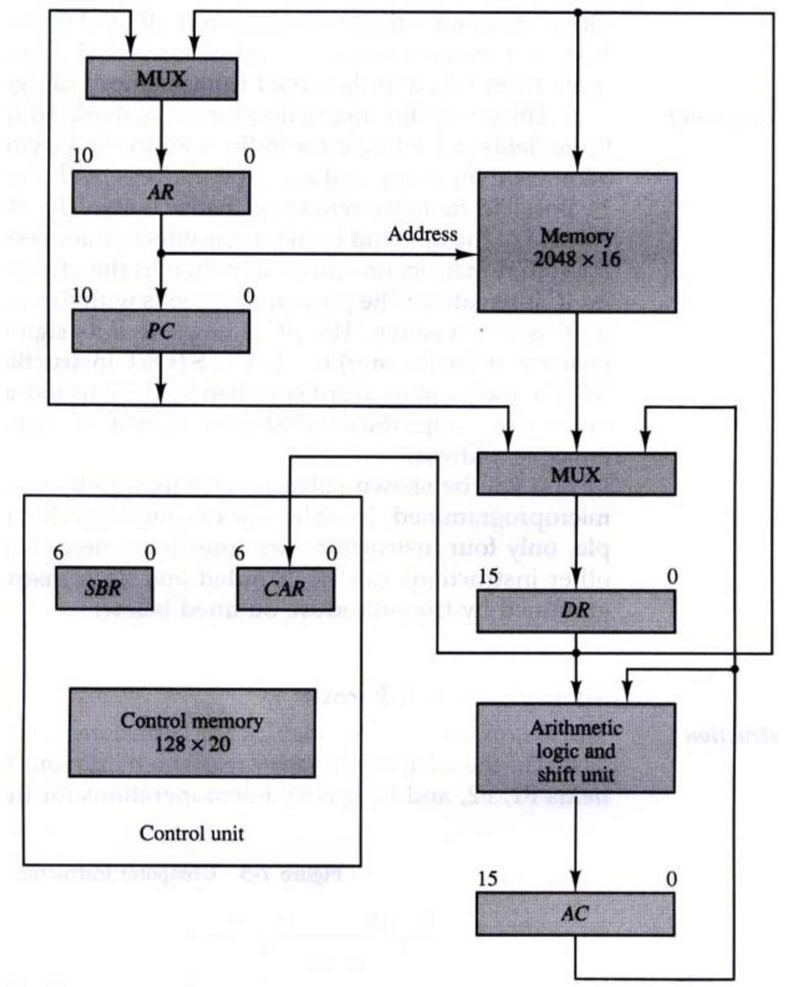
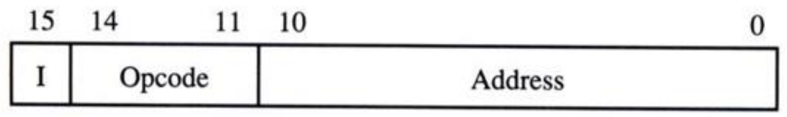
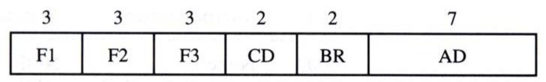


Figure 7-4 Computer hardware configuration.



(a) Instruction format



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

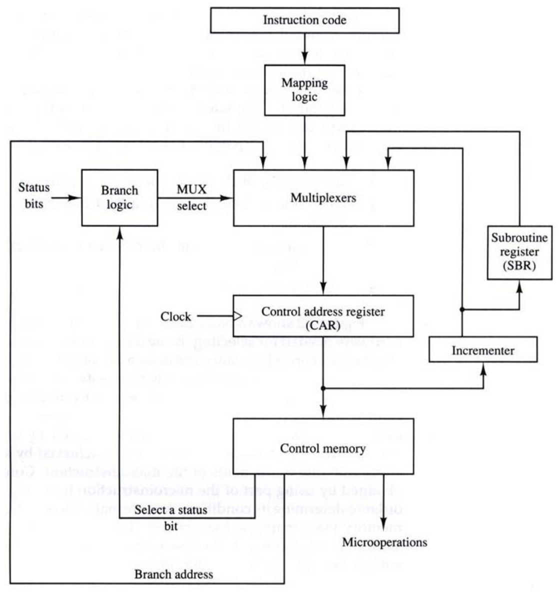
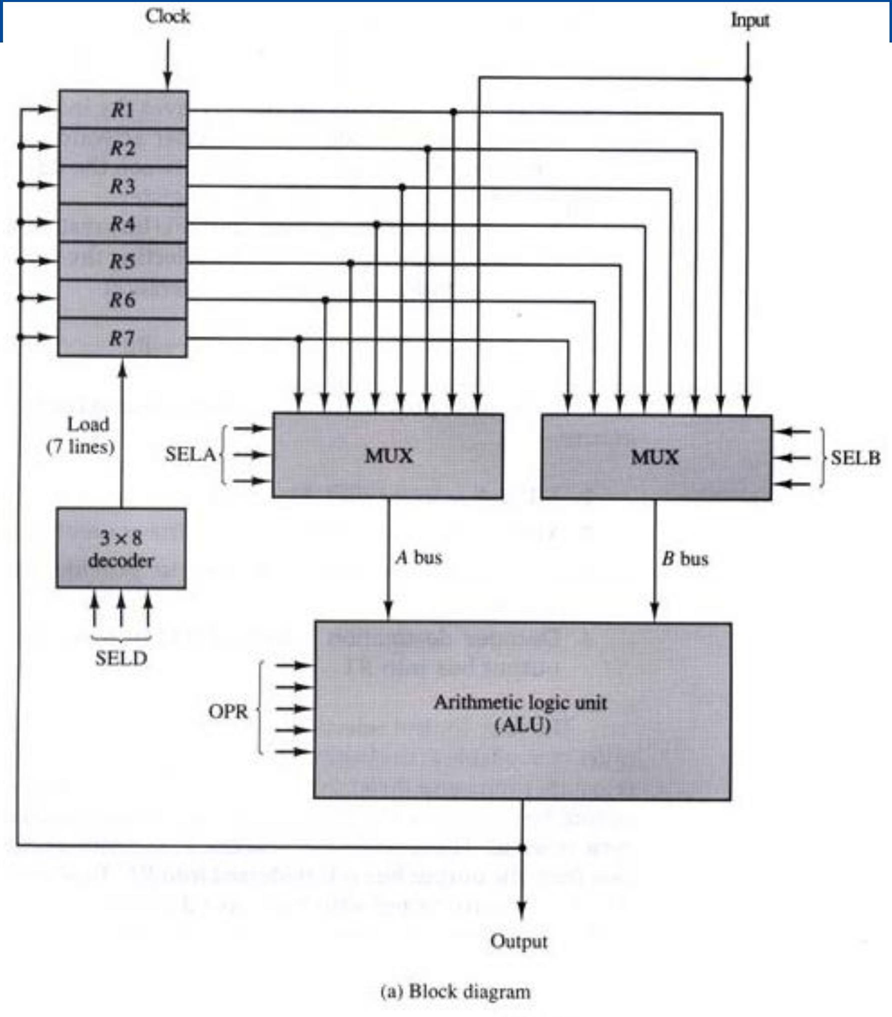
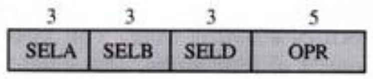


Figure 7-2 Selection of address for control memory.

8.2 General register organization



(a) Block diagram



(b) Control word

Figure 8-2 Register set with common ALU.

8.2 General register organization

- It is efficient and convenient to store the intermediate values in processing registers
- A bus organization for seven CPU registers
- To perform the operation R1 ← R2 + R3
    - MUX A selects R2
    - MUX B selects R3
    - ALU selects arithmetic addition
    - Decoder destination selector selects R1

8.2 General register organization

- Control word
  - 14 binary selection inputs
  - Ex) R1 ← R2 - R3
    - Control word: 010 011 001 00101

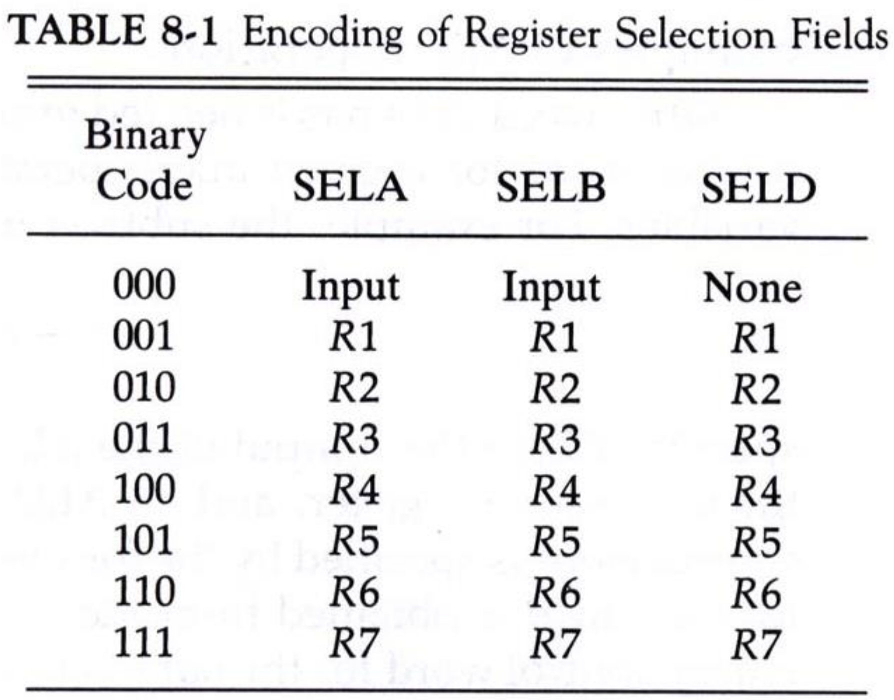


TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7

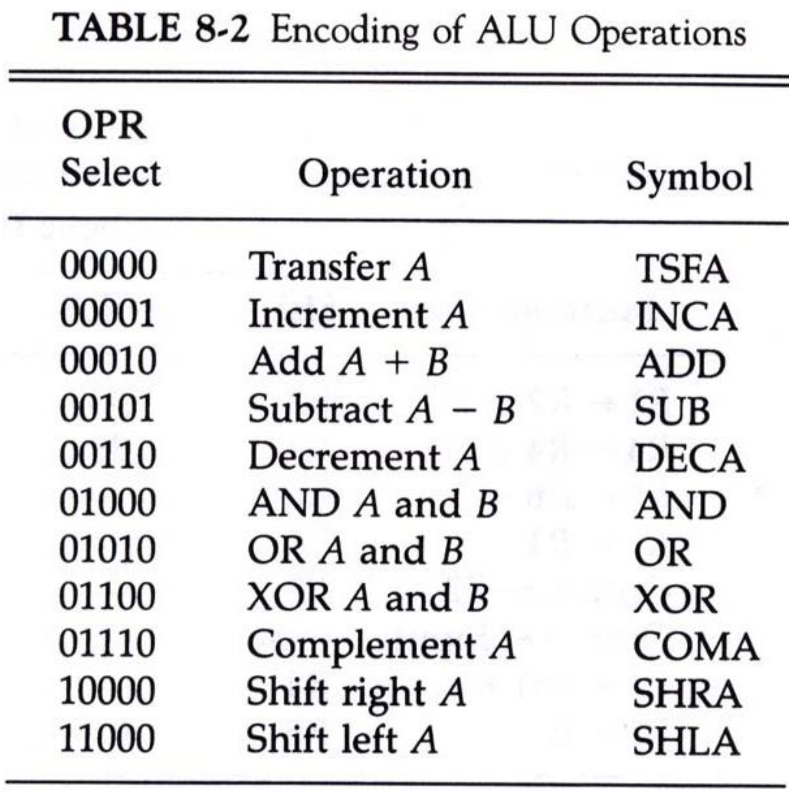


TABLE 8-2 Encoding of ALU Operations

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

8.2 General register organization

- Examples

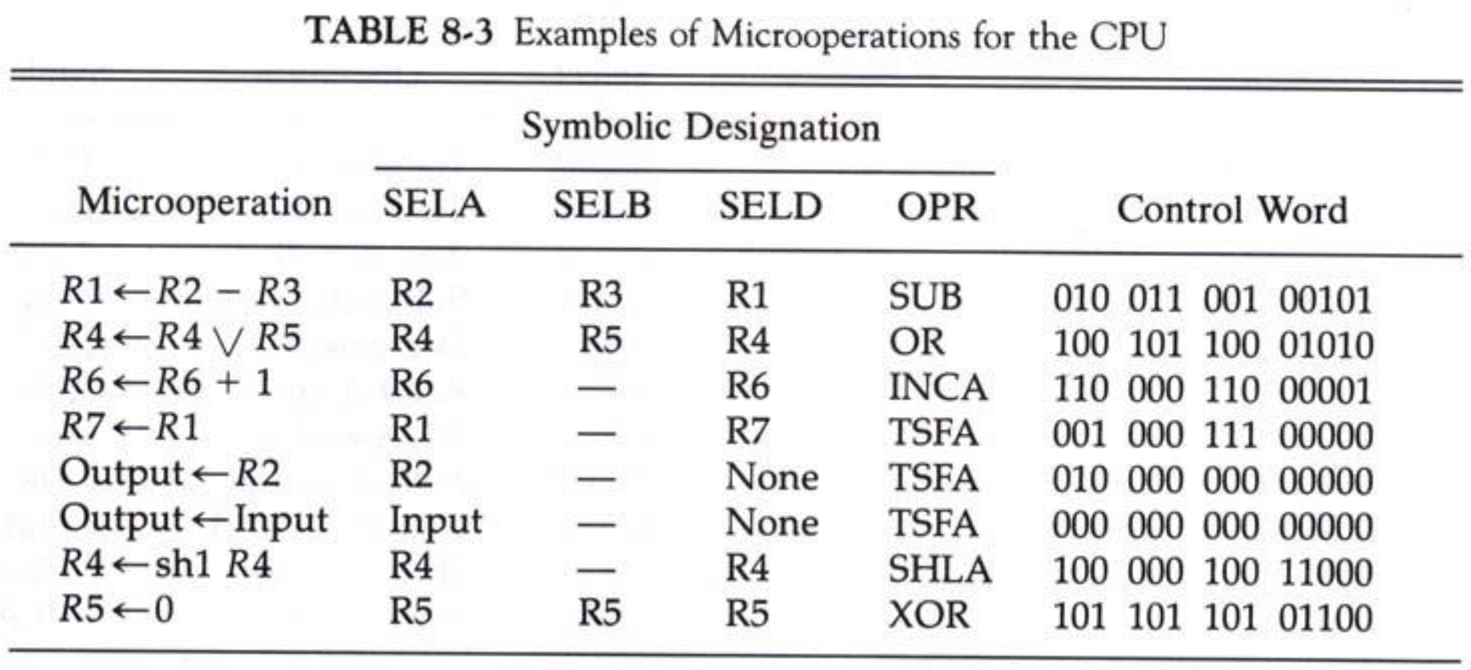


TABLE 8-3 Examples of Microoperations for the CPU

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

8.3 Stack organization

- Stack
  - Last-In First-Out list (LIFO)
  - A memory unit with an address register that can count only (after initial value is loaded into it)
- Register stack
  - Organized as a collection of a finite number of registers
- Memory stack example
  - An organization of a 64-word stack
  - Initially SP is cleaned to 0, EMPTY is set to 1, FULL is cleaned to 0

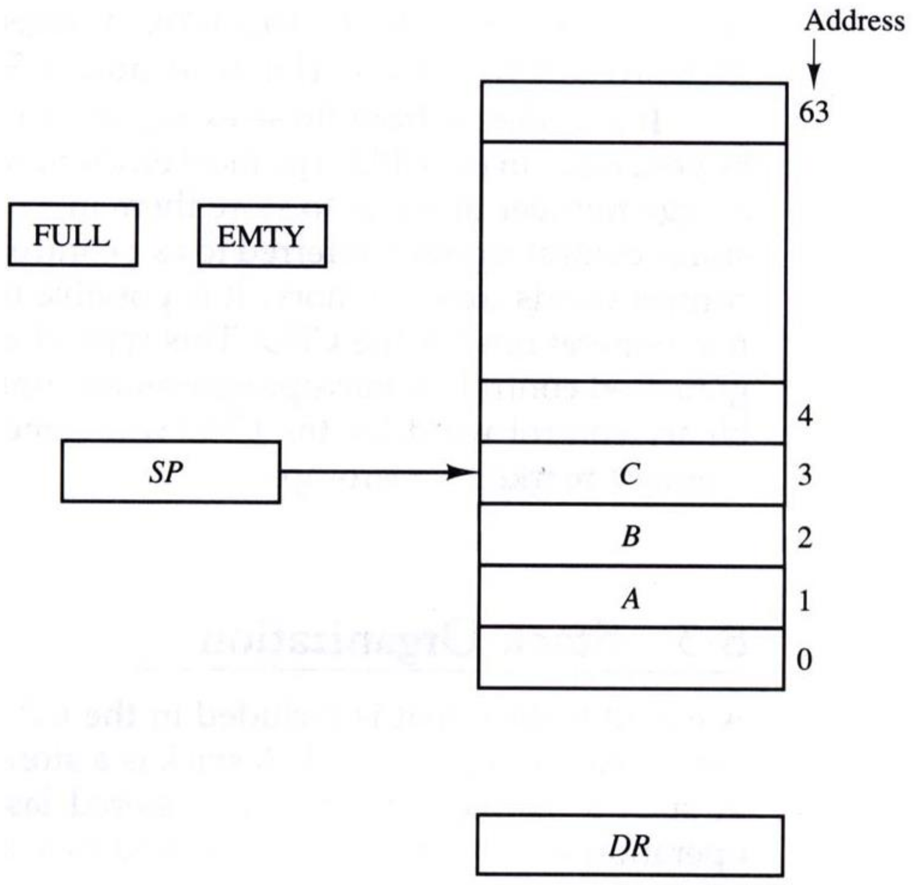


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0
- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0
- The word in address 0 receives the last item in the stack
- Erroneous operation
  - Push when FULL = 1 OR pop when EMPTY = 1

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

Initial condition

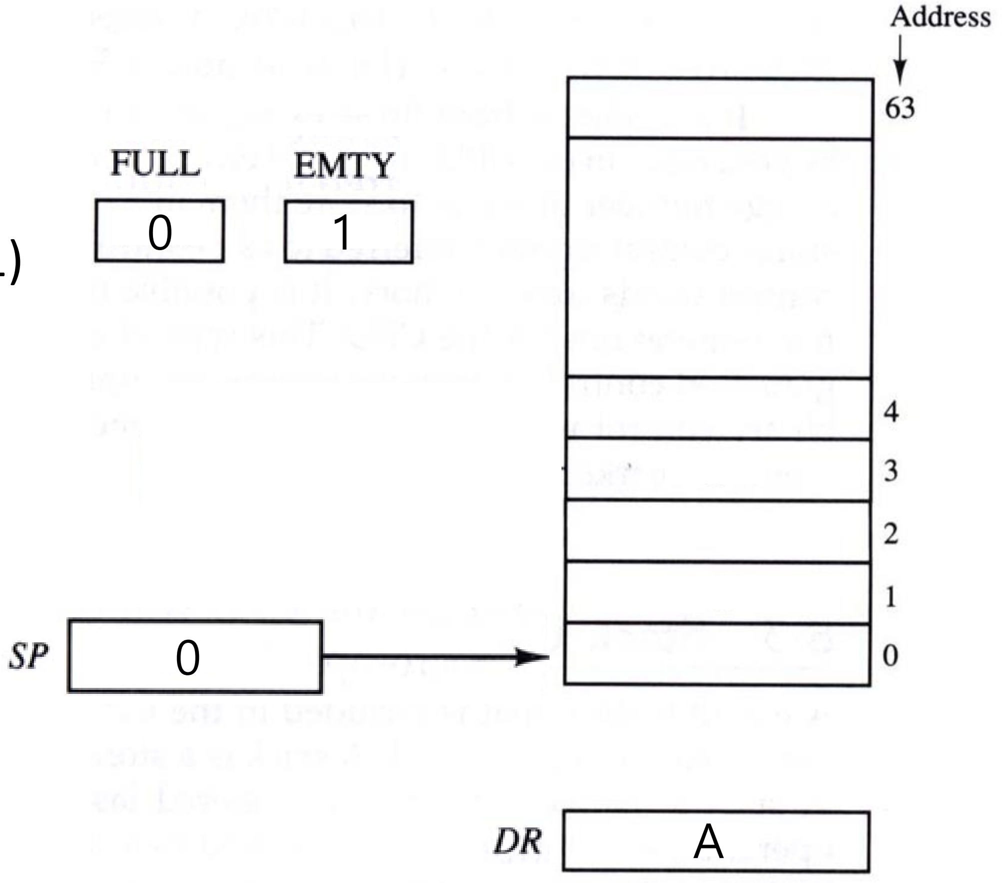


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

PUSH

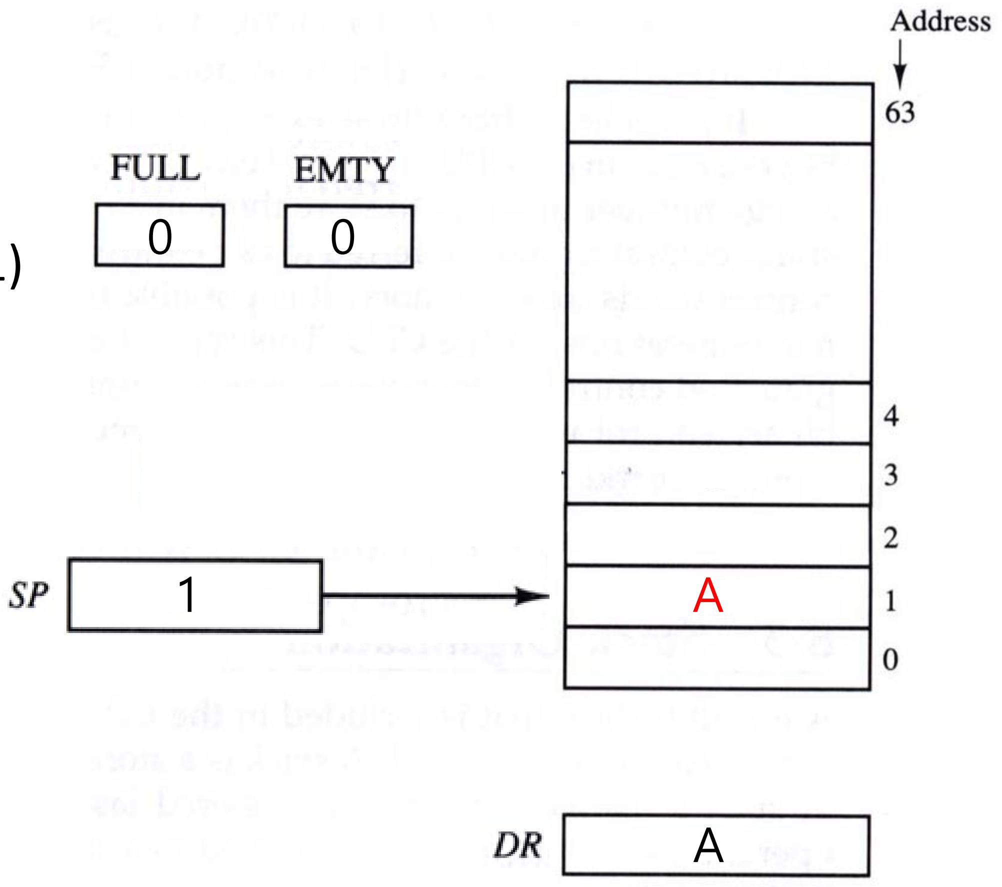


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

PUSH

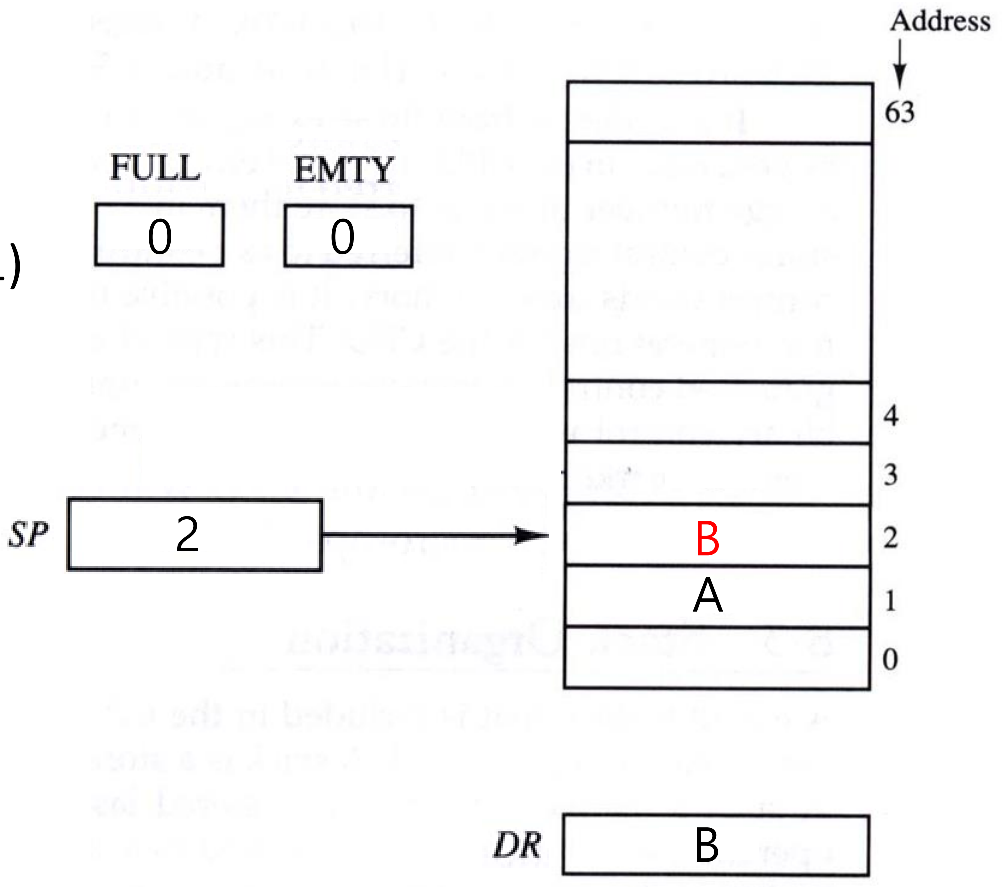


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

PUSH

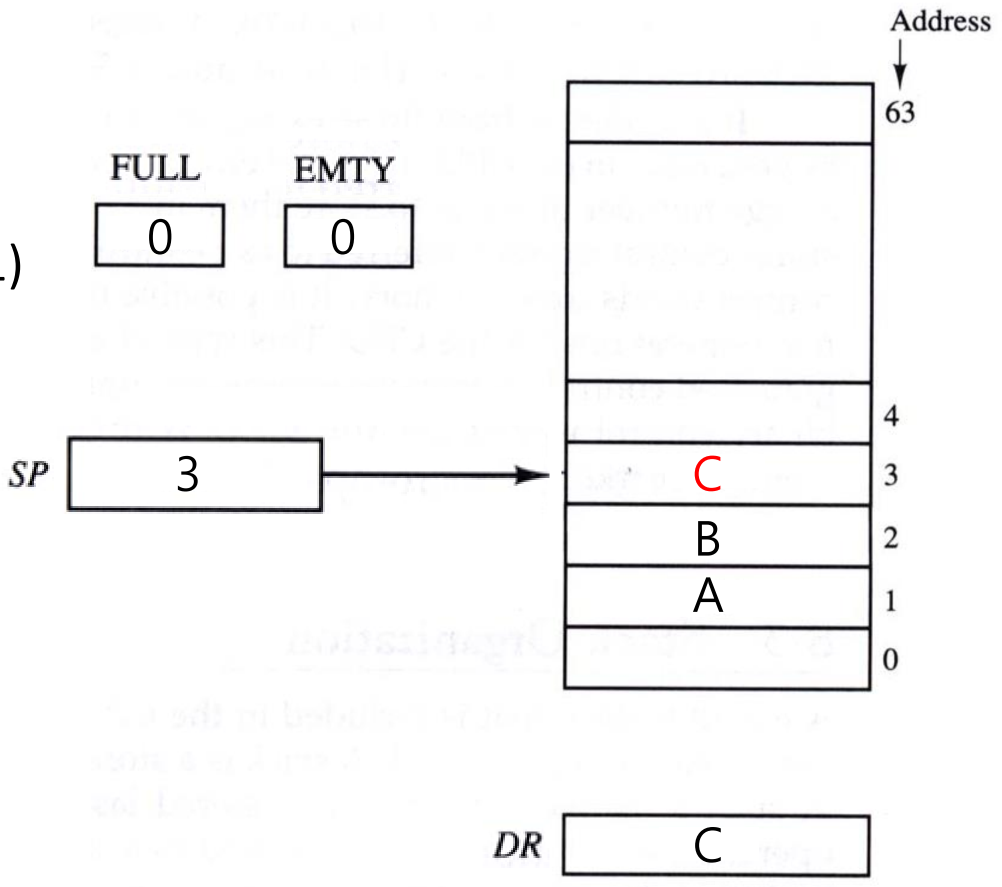


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

PUSH

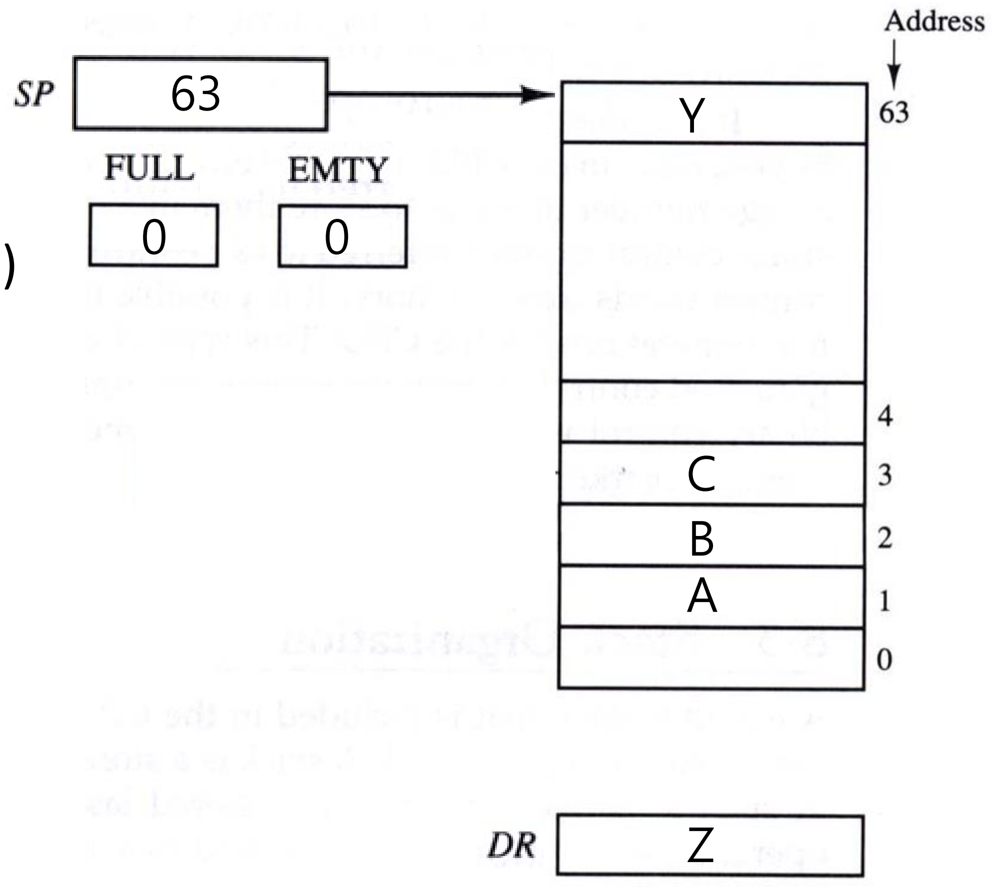


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- PUSH operation
  - SP ← SP + 1
  - M[SP] ← DR
  - IF (SP = 0) then (FULL ← 1)
  - EMPTY ← 0

PUSH

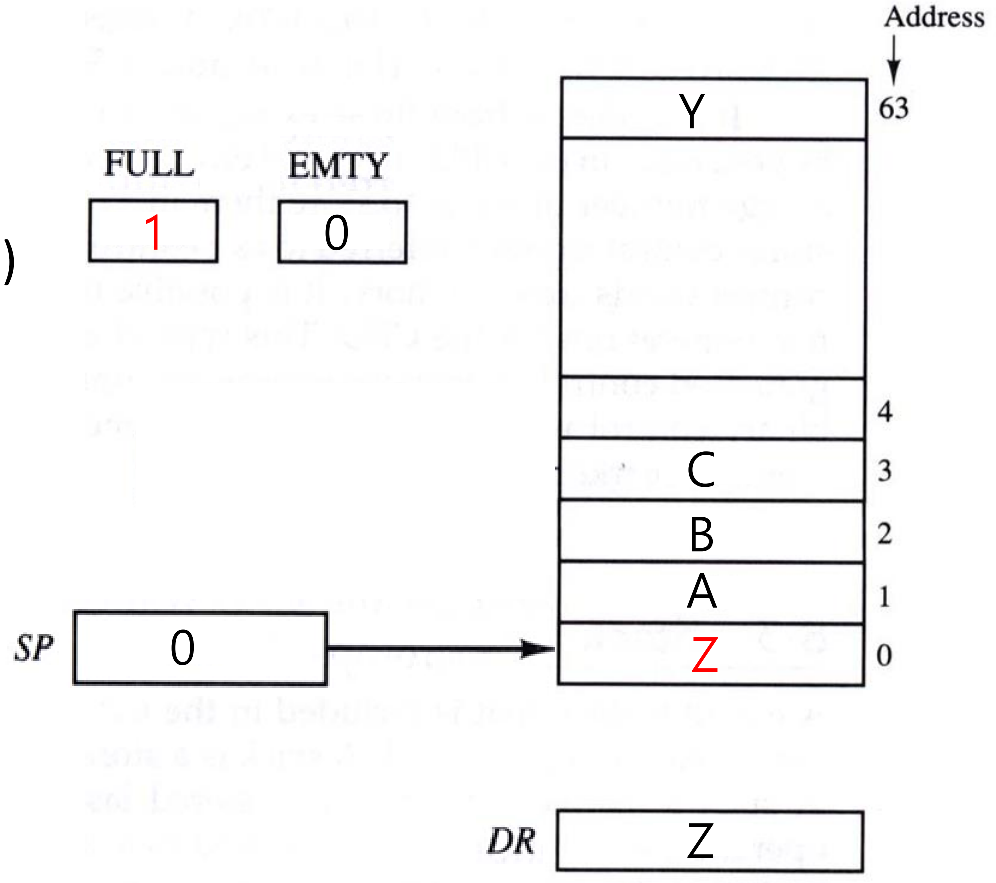


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0

POP

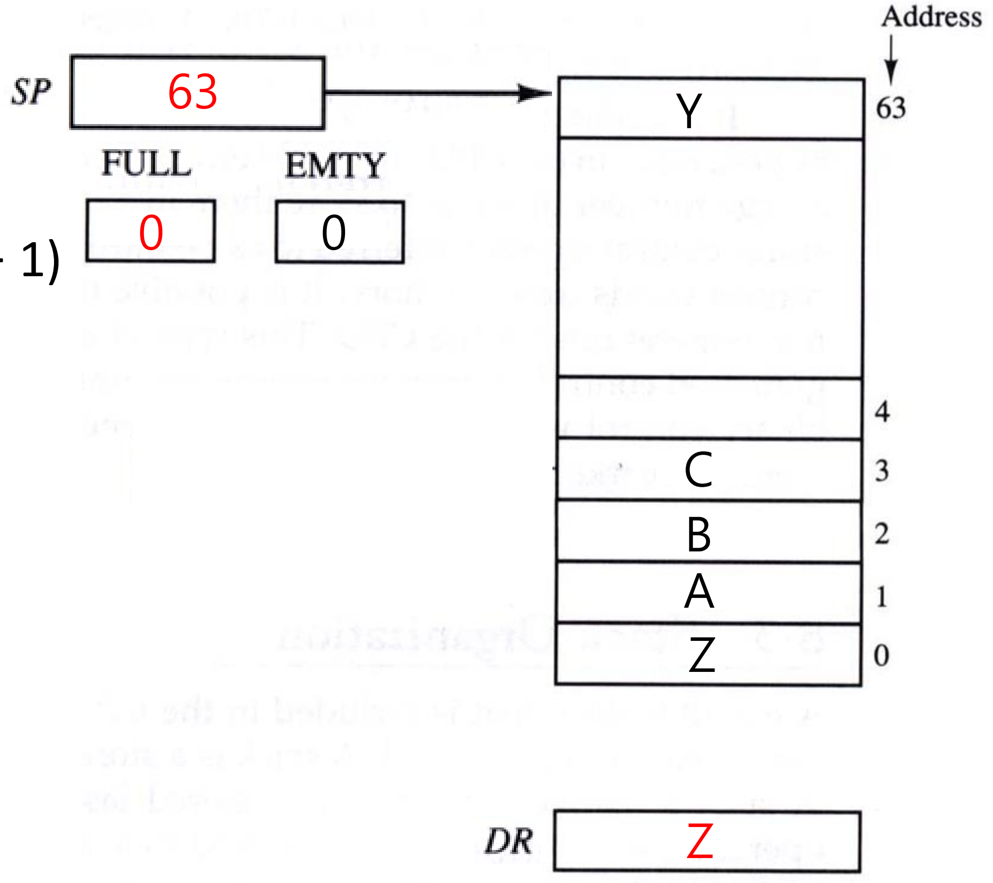


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0

POP

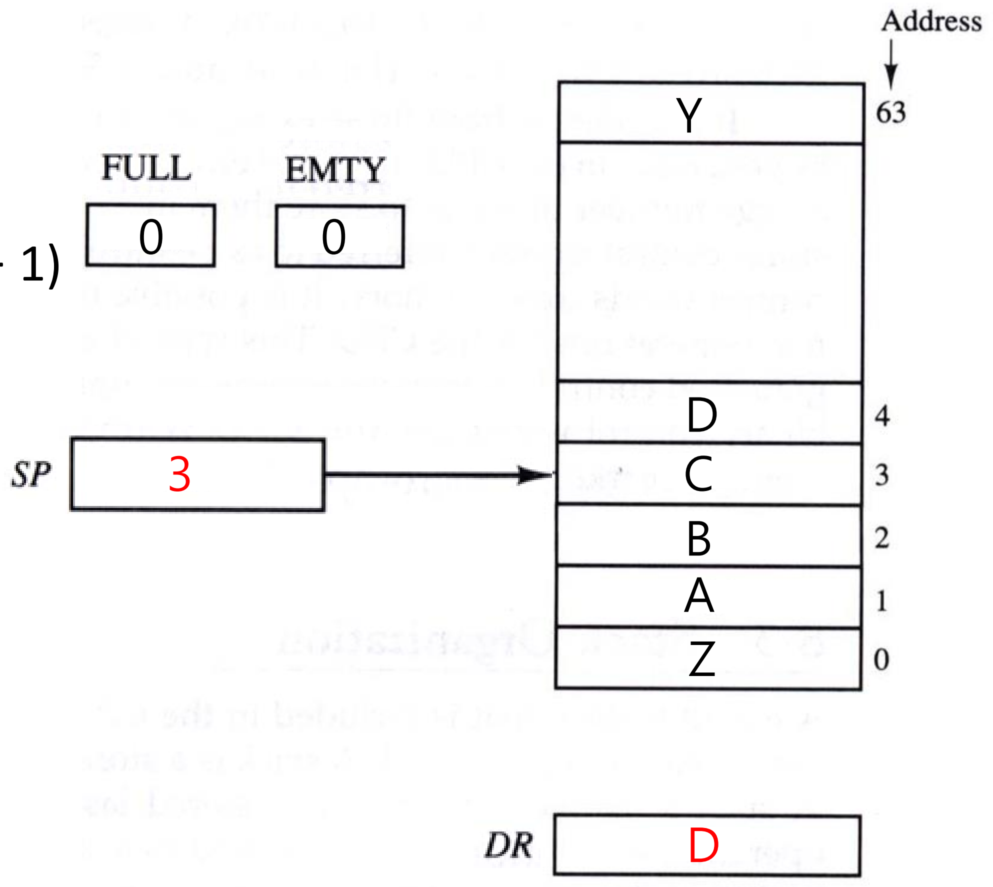


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0

POP

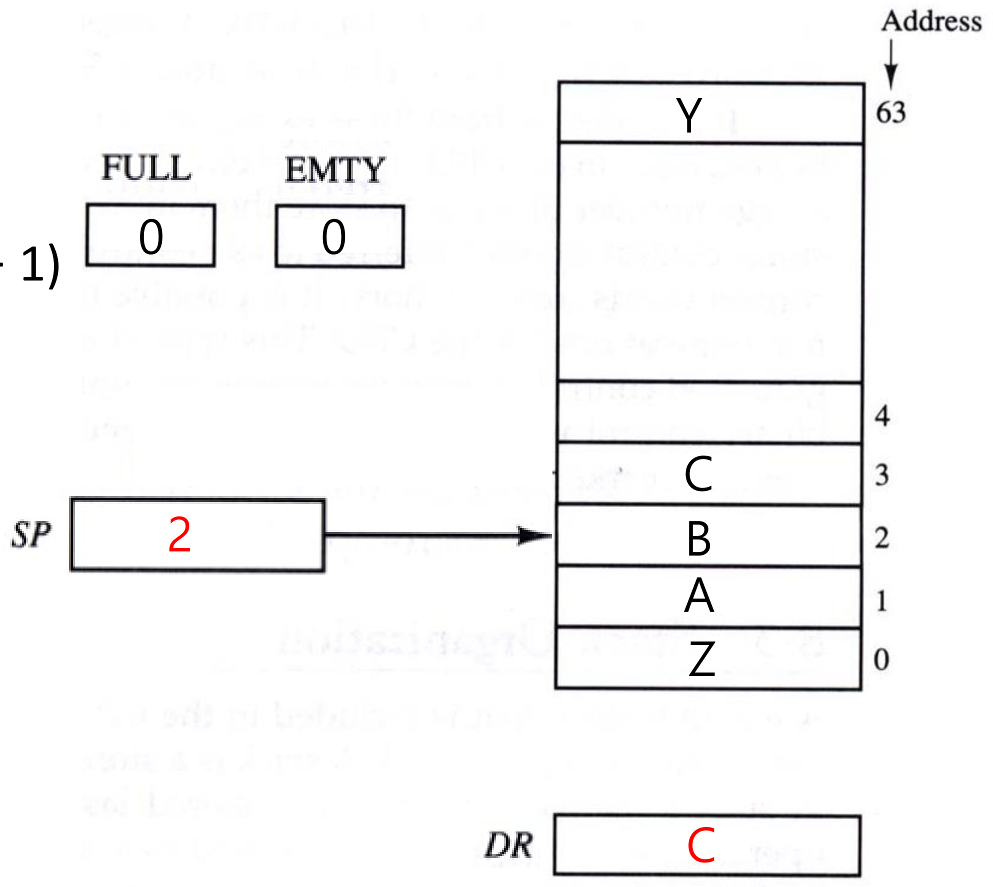


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0

POP

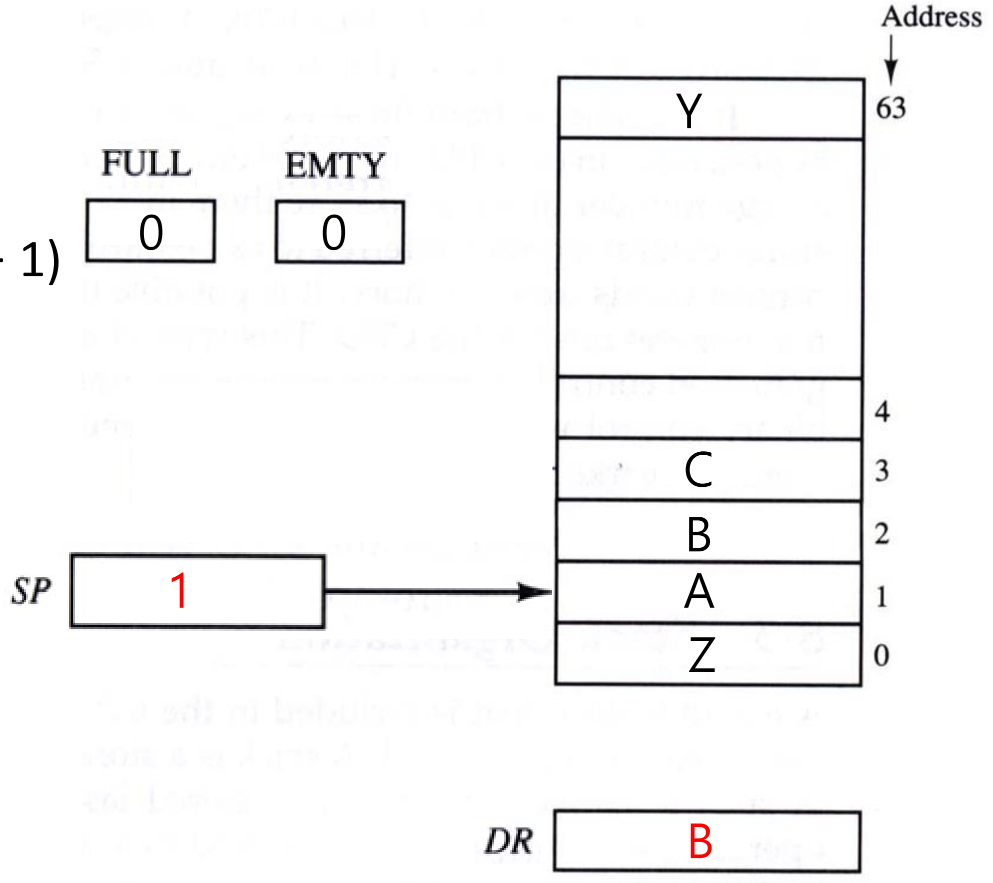


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- POP operation
  - DR ← M[SP]
  - SP ← SP – 1
  - IF (SP = 0) then (EMPTY ← 1)
  - FULL ← 0

POP

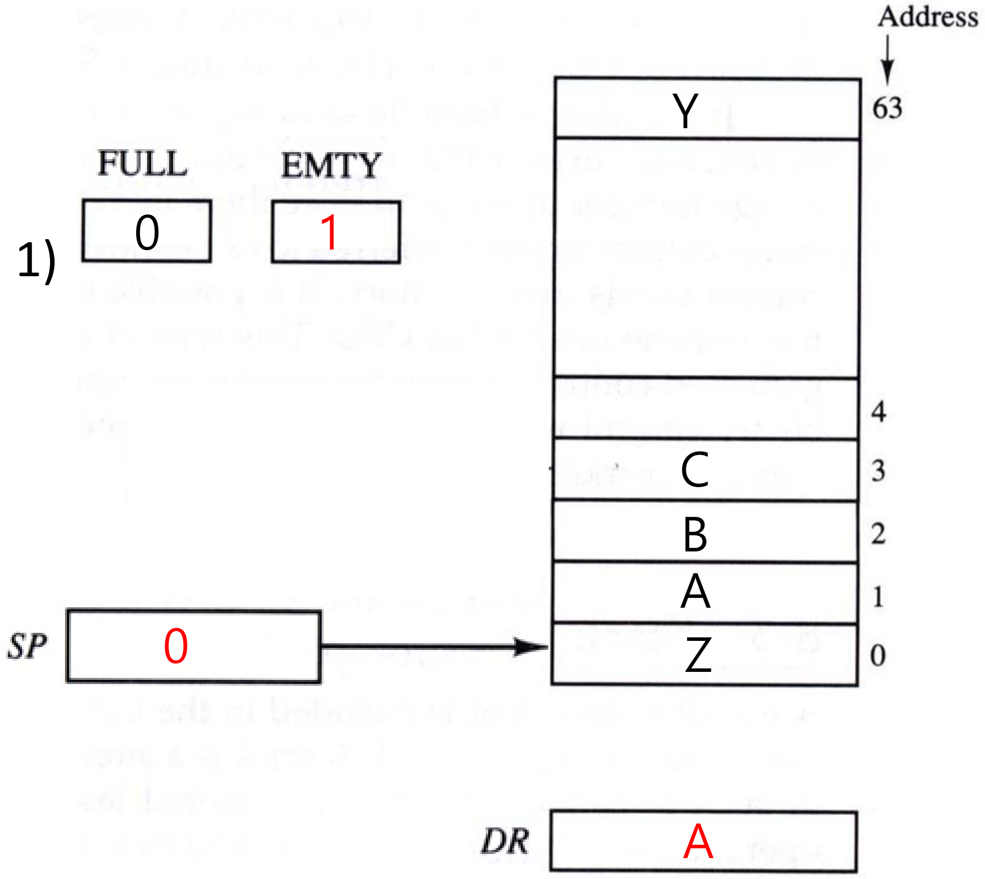


Figure 8-3 Block diagram of a 64-word stack.

8.3 Stack organization

- Memory stack
  - A portion of memory is used as a stack
  - PC is used for instruction fetch
  - AR is used to read operand
  - SP is used to push or pop items into or from the stack
  - The three registers are connected to a common address bus and either one can provide an address for memory
  - The stack limits can be checked by using two processor registers

8.3 Stack organization

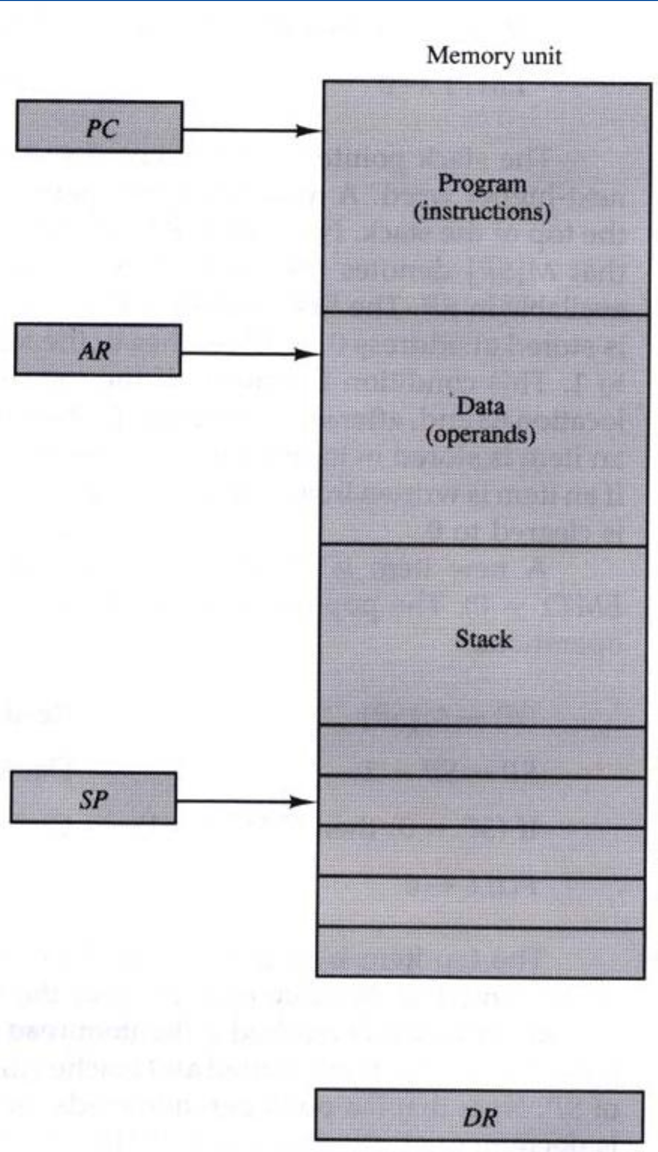


Figure 8-4 Computer memory with program, data, and stack segments.

PUSH

SP ← SP – 1

M[SP] ← DR

POP

DR ← M[SP]

SP ← SP + 1

8.3 Stack organization

- Reverse polish notation (RPN)
  - A + B: infix notation
  - + AB: prefix or polish notation
  - AB +: postfix or reverse polish notation (RPN)
- RPN is in a form suitable for stack manipulation
- When an operator is reached, perform the operation with the two operands found at the left side of the operator
- Example of evaluation
  - AB*CD*+
  - (A*B)CD*+
  - (A*B)(C*D)+
  - (A*B)+(C*D)

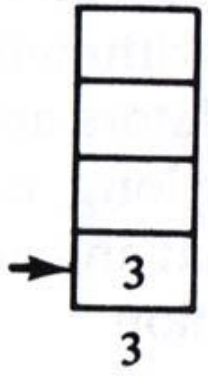
8.3 Stack organization

- Conversion to RPN
  - Ex) (A+B)*[C*(D+E)+F] → AB+ DE+ C* F+ *
- Stack operation of arithmetic expression
  - The operands are pushed into the stack
  - For an operator, two operands are popped and the operation is performed, and the result is pushed into the stack

8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

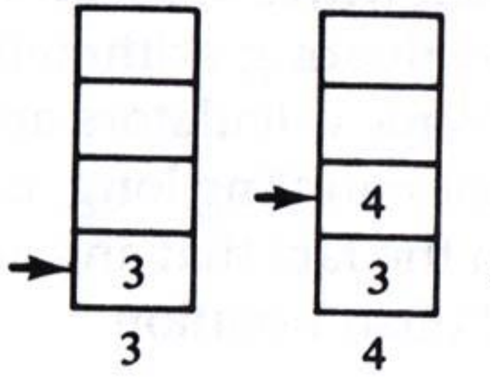
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

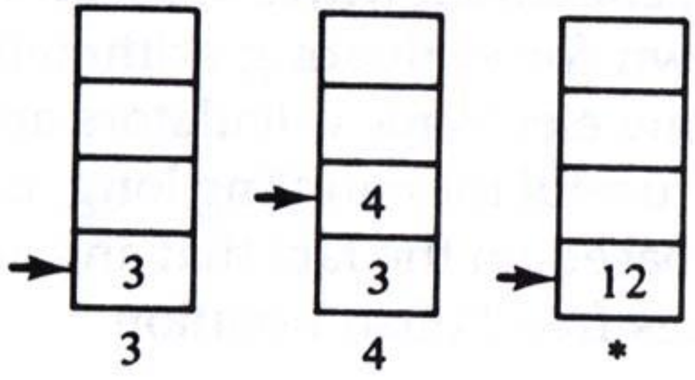
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

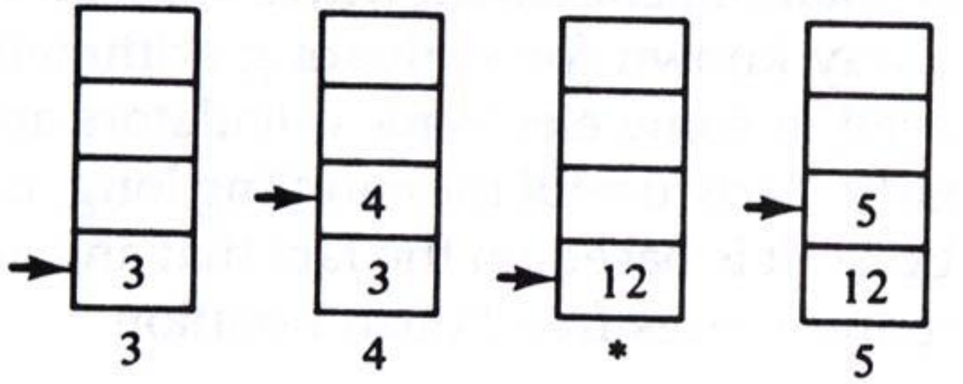
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

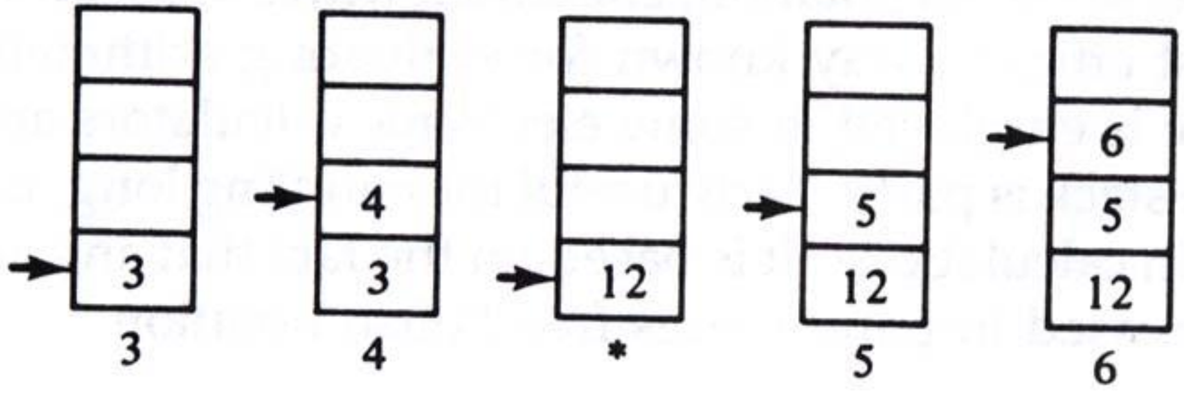
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

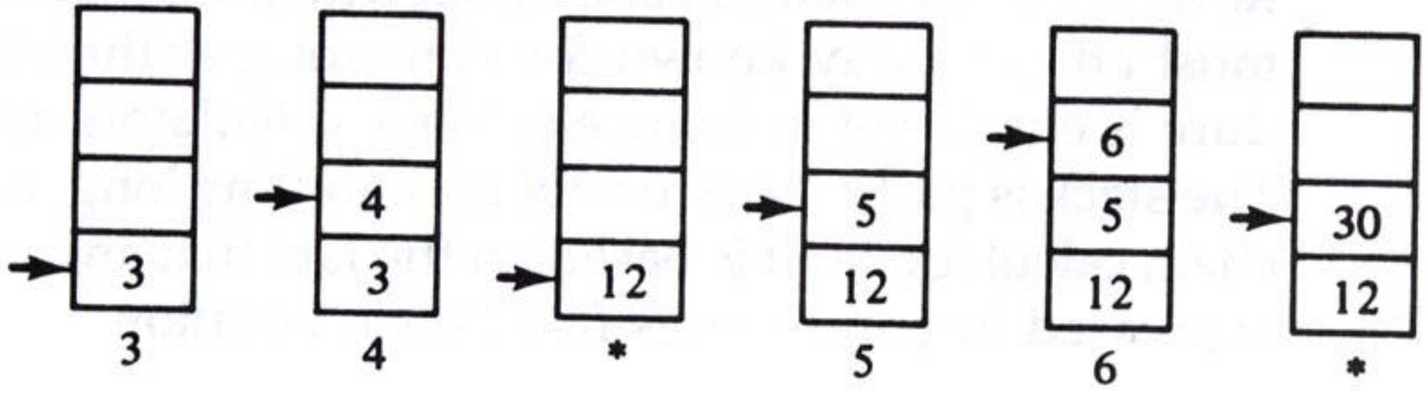
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

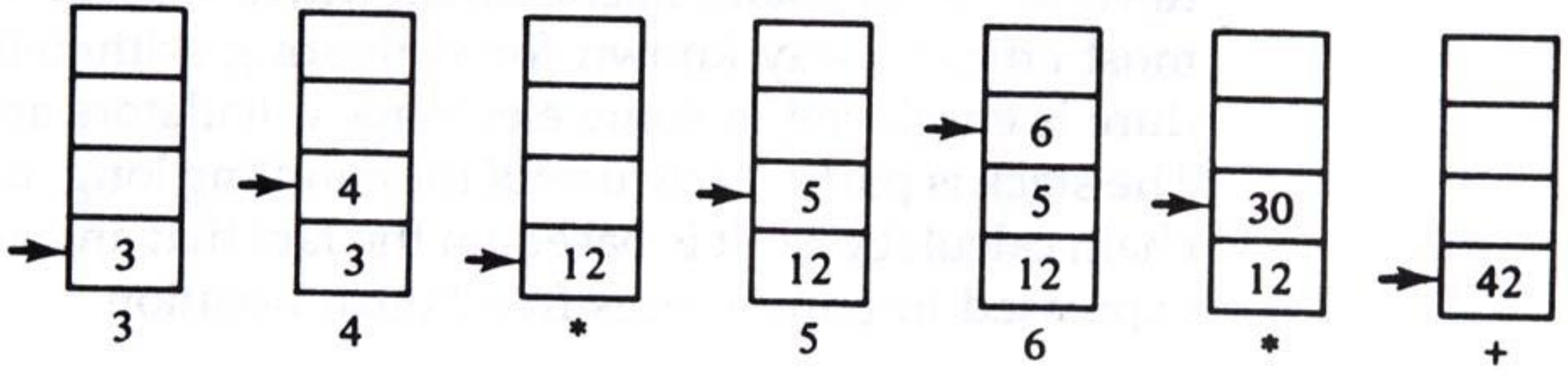
Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.3 Stack organization

- Example of stack operation of arithmetic expression
  - (3 * 4) + (5 * 6)
  - RPN: 34 * 56 * +

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



8.4 Instruction formats

- The most common fields in instruction formats
  - Operation code field
  - Address field (register or memory)
  - Mode field
- Type of CPU organizations
  - Single accumulator organization
  - General register organization
  - Stack organization

8.4 Instruction formats

- Instruction format for single accumulator organization
  - ADD X
- Instruction format for general register organization
  - ADD R1, R2, R3; R1 ← R2 + R3
  - ADD R1, R2 ; R1 ← R1 + R2
  - MOV R1, R2
  - ADD R1, X
- Instruction format for stack organization
  - PUSH X
  - ADD

8.4 Instruction formats

- The number of addresses
  - Ex) X = (A + B) * (C + D)

- Three address instructions
  - Short program, long 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

8.4 Instruction formats

- Two address instructions
  - Most common

  - MOV R1, A ; $R1 \leftarrow M[A]$
  - ADD R1, B ; $R1 \leftarrow R1 + M[B]$
  - MOV R2, C ; $R2 \leftarrow M[C]$
  - ADD R2, D ; $R2 \leftarrow R2 + M[D]$
  - MUL R1, R2 ; $R1 \leftarrow R1 * R2$
  - MOV X, R1 ; $M[X] \leftarrow R1$

8.4 Instruction formats

- One address instructions
  - Implied accumulator register
  - 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

8.4 Instruction formats

- Zero address instruction
  - Stack organized computer

  - PUSH A ; TOS $\leftarrow$ A
  - PUSH B ; TOS $\leftarrow$ B
  - ADD ; TOS $\leftarrow$ (A + B)
  - PUSH C ; TOS $\leftarrow$ C
  - PUSH D ; TOS $\leftarrow$ D
  - ADD ; TOS $\leftarrow$ (C + D)
  - MUL ; TOS $\leftarrow$ (C + D) * (A + B)
  - POP X ; M[X] $\leftarrow$ TOS

8.4 Instruction formats

- RISC instruction
  - Restricted to use of load and store instructions when communicating between memory and CPU
  - All other instructions are executed within registers

  - LOAD R1, A ; $R1 \leftarrow M[A]$
  - LOAD R2, B ; $R2 \leftarrow M[B]$
  - LOAD R3, C ; $R3 \leftarrow M[C]$
  - LOAD R4, D ; $R4 \leftarrow M[D]$
  - ADD R1, R1, R2 ; $R1 \leftarrow R1 + R2$
  - ADD R3, R3, R4 ; $R3 \leftarrow R3 + R4$
  - MUL R1, R1, R3 ; $R1 \leftarrow R1 * R3$
  - STORE X, R1 ; $M[X] \leftarrow R1$

8.5 Addressing modes

- Purpose of addressing mode techniques
  - To give programming versatility to the user (e.g., pointers, indexing of data, counters of loop control, program relocation)
  - To reduce the number of bits in the addressing field of the instruction
- Implied mode
- immediate mode
- Register mode, register indirect mode
- Auto-increment or auto-decrement mode
- Direct address mode, indirect address mode
- Relative address mode, indexed addressing mode, base register address mode
  - Effective address = address part of instruction + content of CPU register
  - Program counter (PC), index register (XR), base register

8.5 Addressing modes

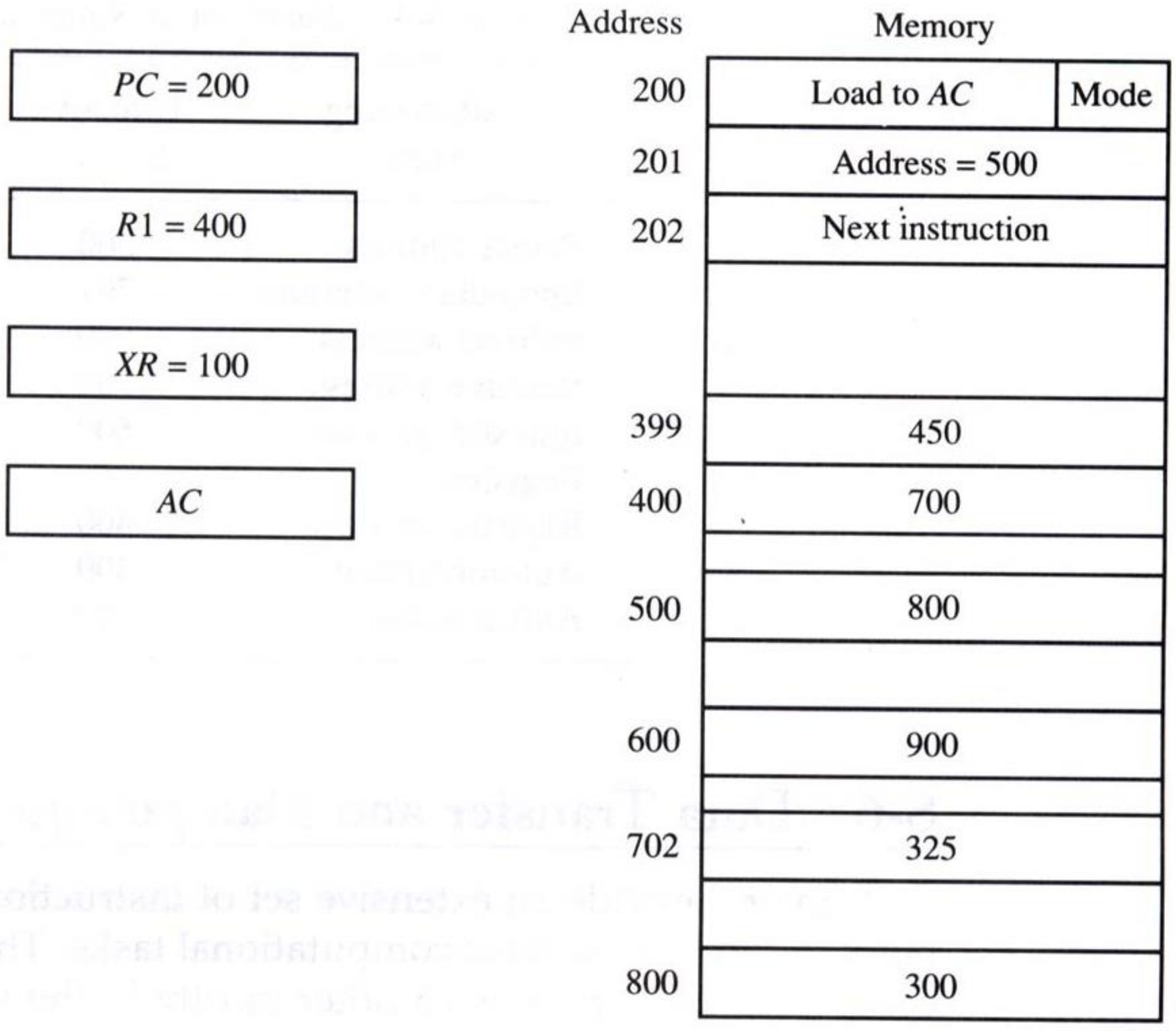


Figure 8-7 Numerical example for addressing modes.

8.5 Addressing modes

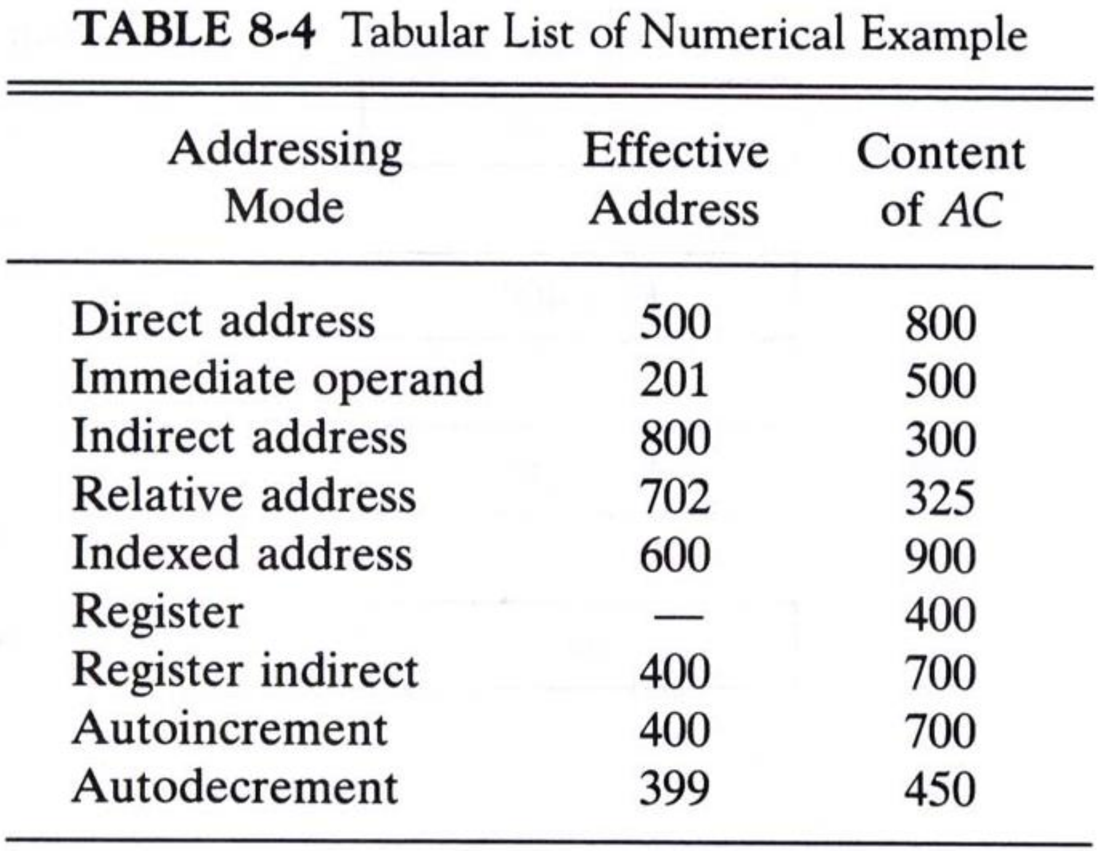


TABLE 8-4 Tabular List of Numerical Example

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

8.6 Data transfer & manipulation

- Three categories of instructions
  - Data transfer instructions
  - Data manipulation instructions
  - Program control instructions
- Data transfer instructions
  - Load, store, exchange, load immediate, etc.

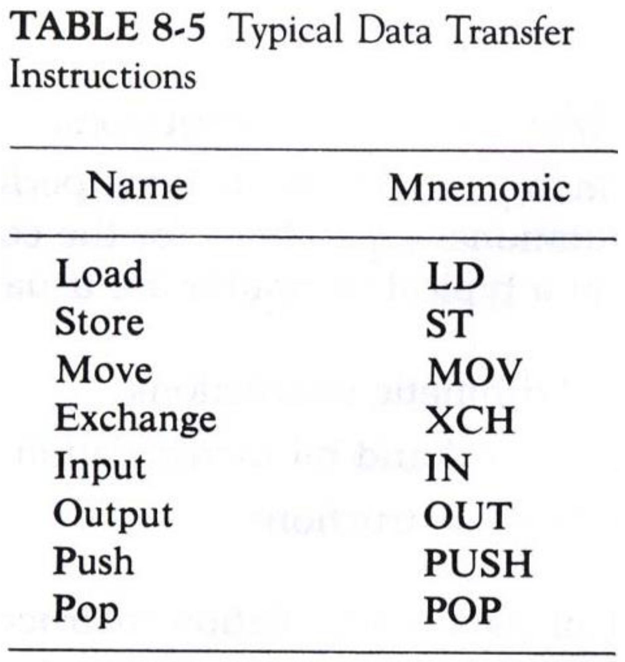


TABLE 8-5 Typical Data Transfer Instructions

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

8.6 Data transfer & manipulation

- Data transfer instructions
  - Used with different addressing modes

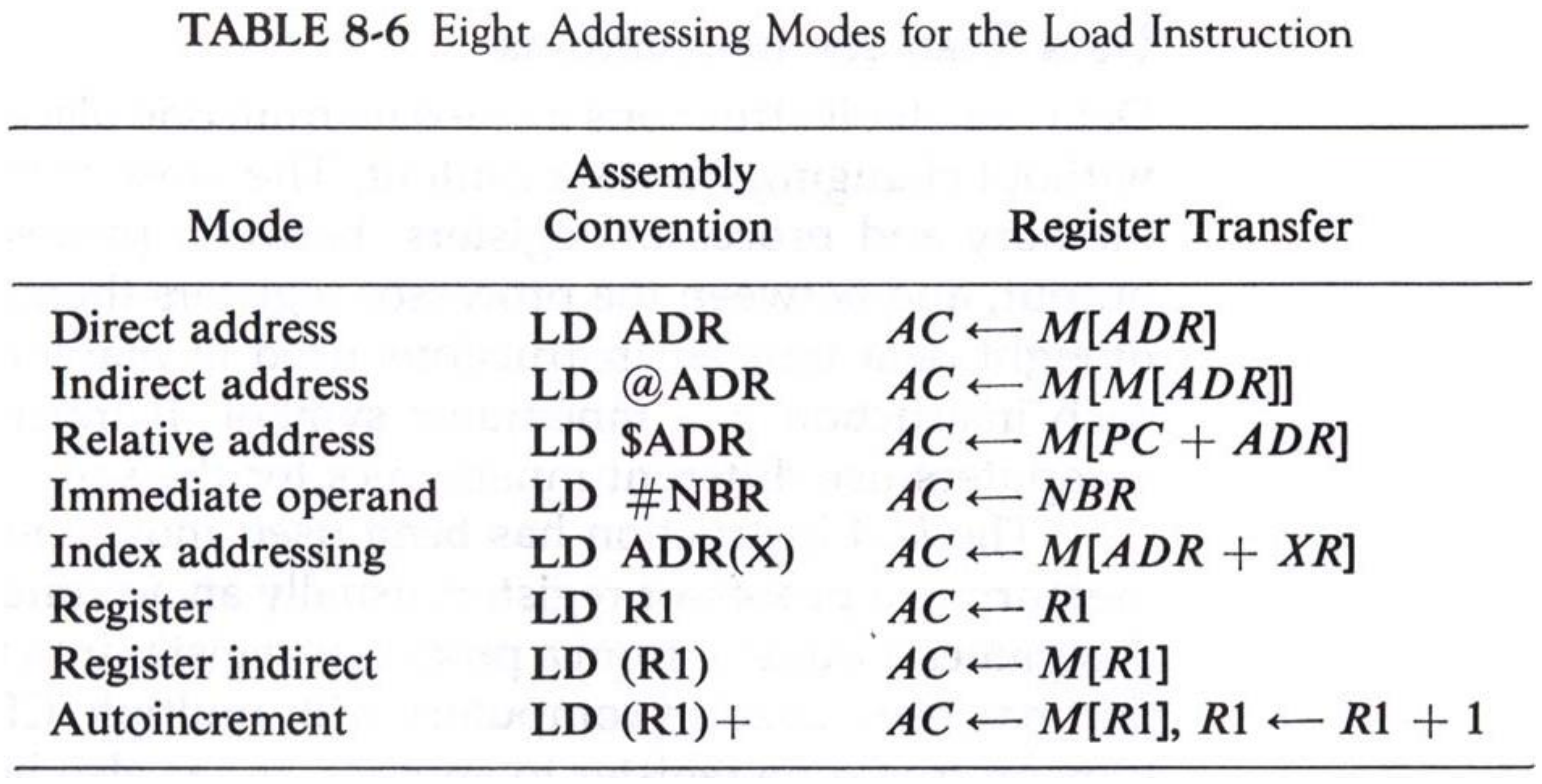


TABLE 8-6 Eight Addressing Modes for the Load Instruction

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)+	$AC \leftarrow M[R1],\ R1 \leftarrow R1 + 1$

8.6 Data transfer & manipulation

- Data manipulation instructions
  - Arithmetic instructions
    - Add, subtract, multiply, etc. (for different types of data)

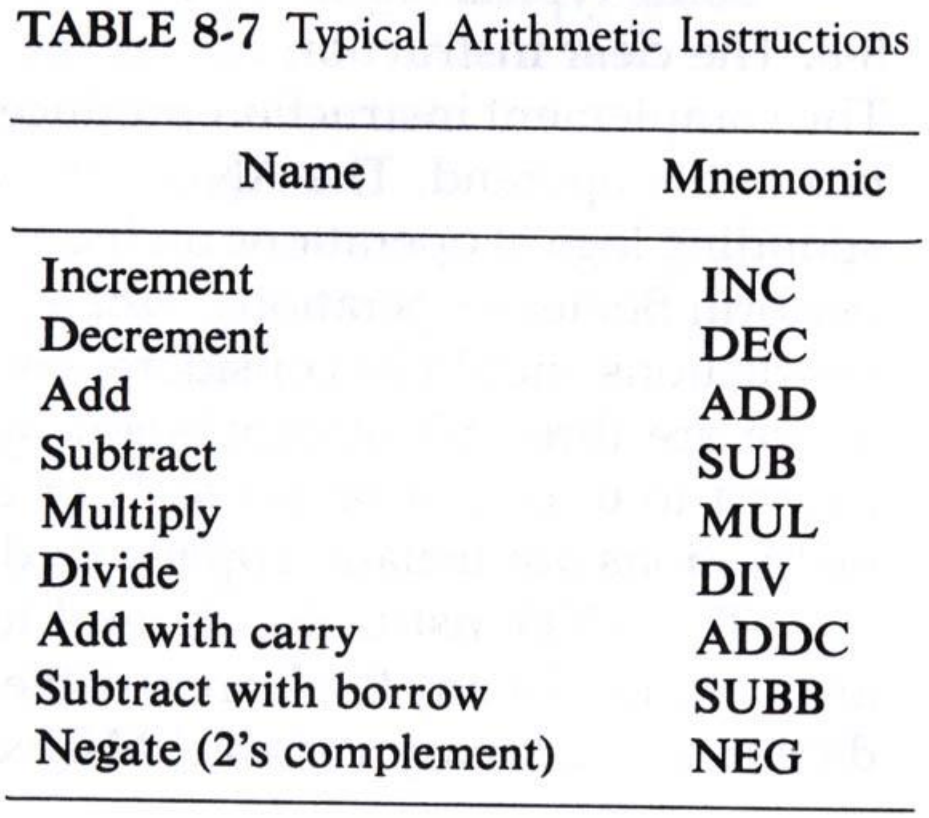


TABLE 8-7 Typical Arithmetic Instructions

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

8.6 Data transfer & manipulation

- Data manipulation instructions
  - Logical and bit manipulation instructions
    - AND, OR, clear, complement, etc.

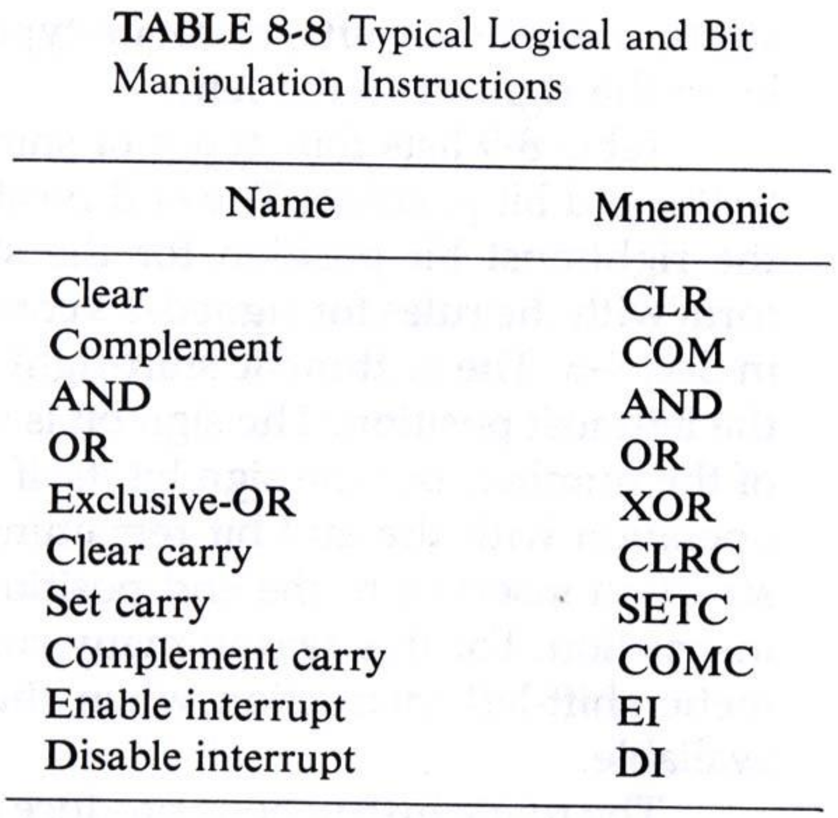


TABLE 8-8 Typical Logical and Bit Manipulation Instructions

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

8.6 Data transfer & manipulation

- Data manipulation instructions
  - Shift instructions
    - Logical shift, arithmetic shift, rotate, rotate through carry

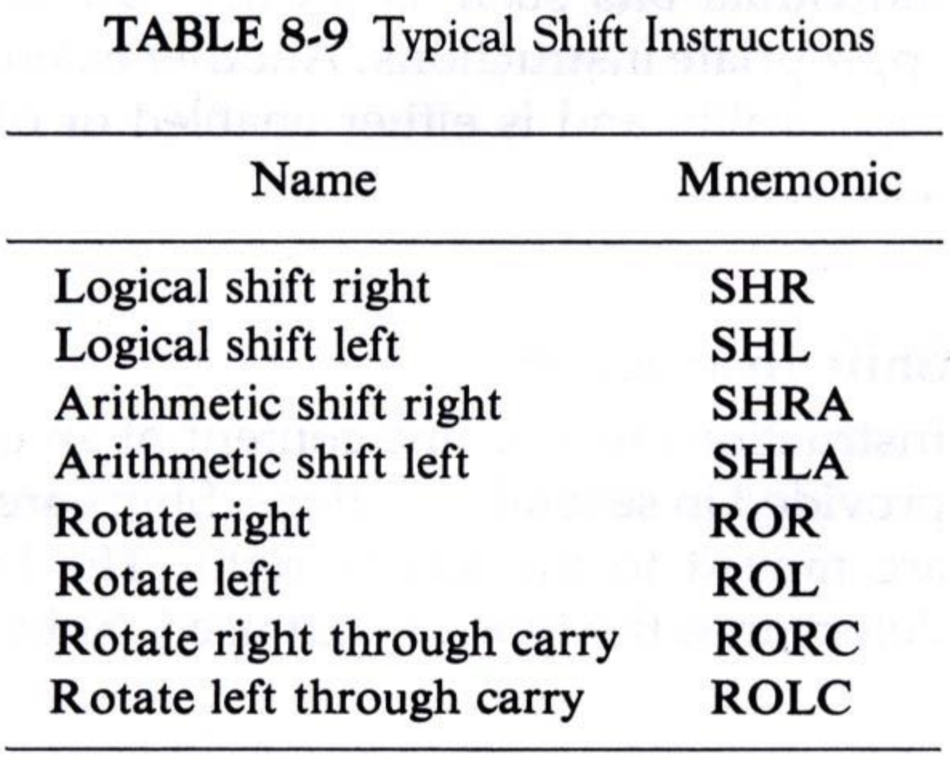


TABLE 8-9 Typical Shift Instructions

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

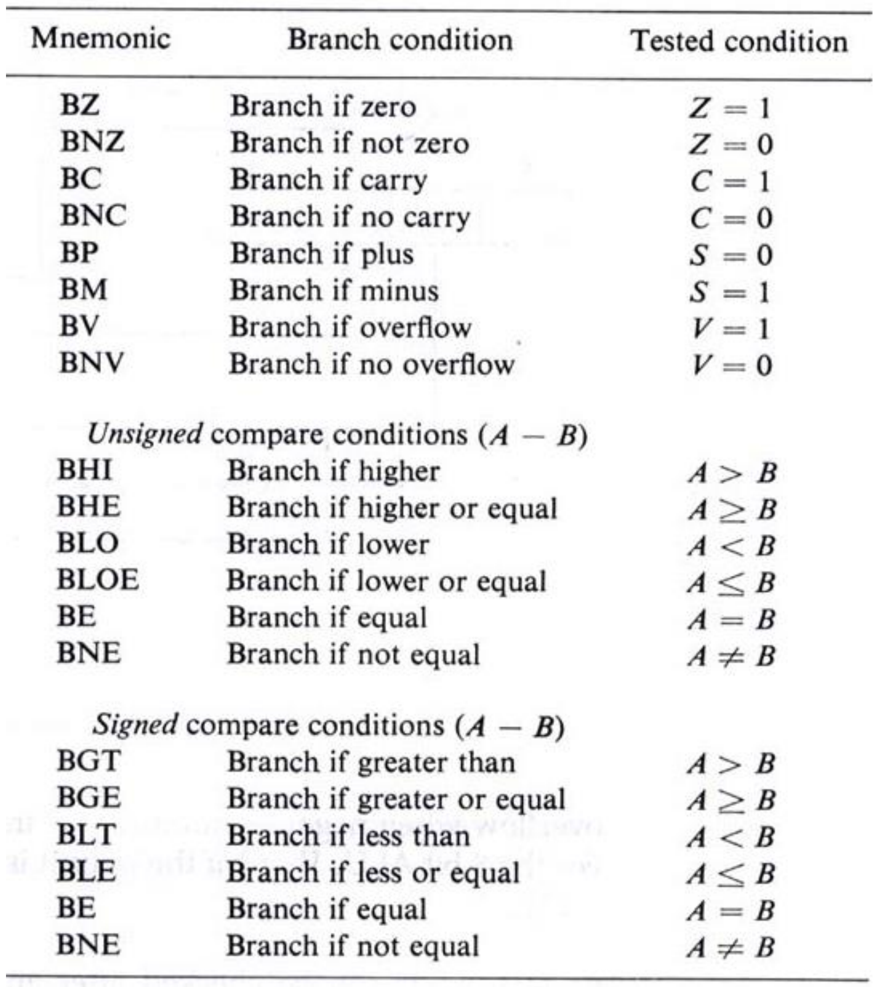
8.7 Program control

- Program control type instructions may change the value in the program counter
  - (Conditional or unconditional) branch, skip, call, return, etc.
- Status bit conditions
  - Compare and test instructions set certain status bits
  - C (carry) is set to 1 if the end carry C_8 is 1
  - S (sign) is set to 1 if the highest order bit F_7 is 1
  - Z (zero) is set to 1 if the output of ALU contains all 0's
  - V (overflow) is set to 1 if exclusive-OR of C_7 and C_8 is equal to 1

8.7 Program control

- Conditional branch instructions

TABLE 8-11 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	$A \neq B$

8.7 Program control

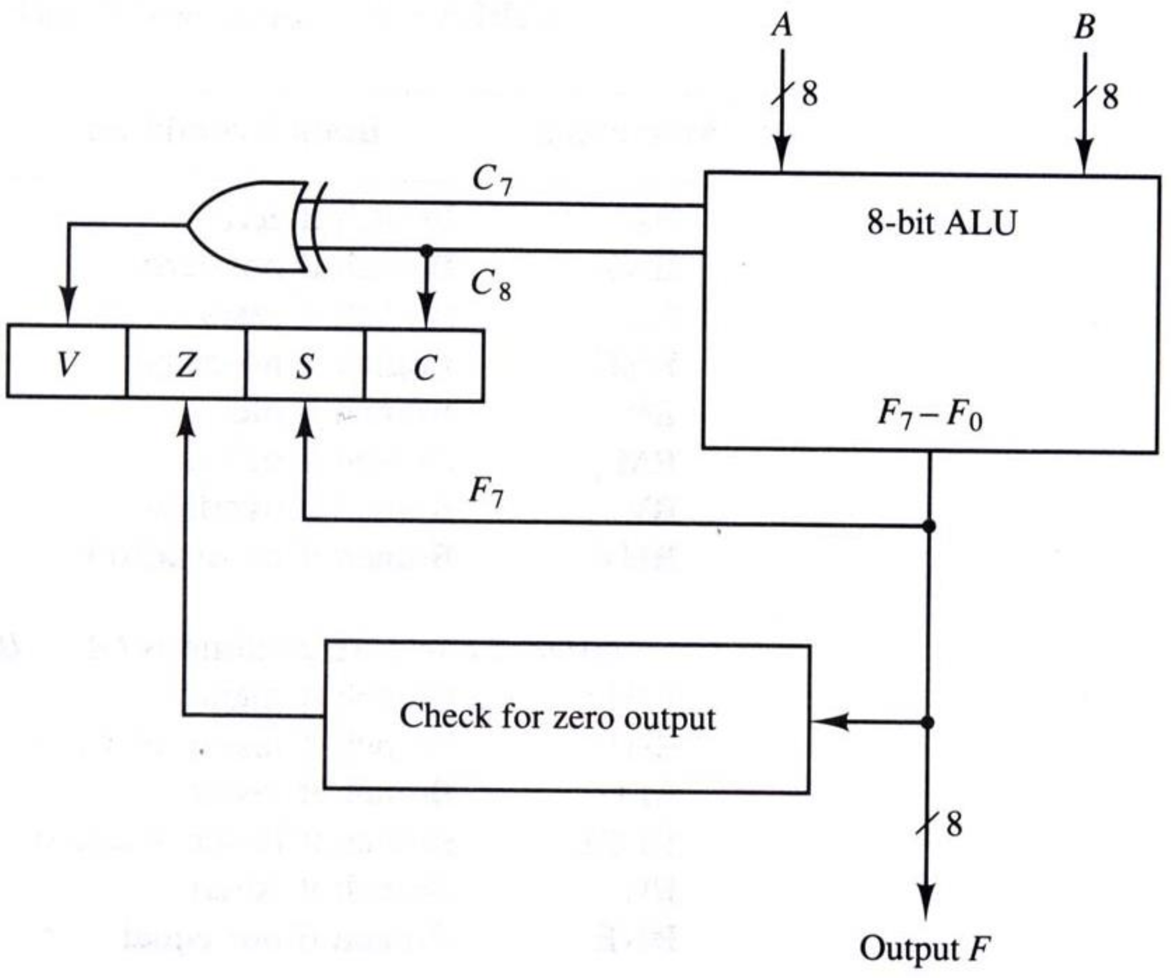


Figure 8-8 Status register bits.

8.7 Program control

- Subroutine call and return
  - A self-contained sequence of instructions
  - Can be called to perform its function many times at various point in main program
  - *Call subroutine, jump to subroutine, branch to subroutine, branch and save address*
  - (1) Address of the next instruction in PC (return address) is stored in a temporary location
  - (2) Last instruction of every subroutine transfers the return address to PC - *return from subroutine*
  - Return address can be stored in the first memory location of the subroutine, a fixed location in memory, a processor register, a memory stack, etc.

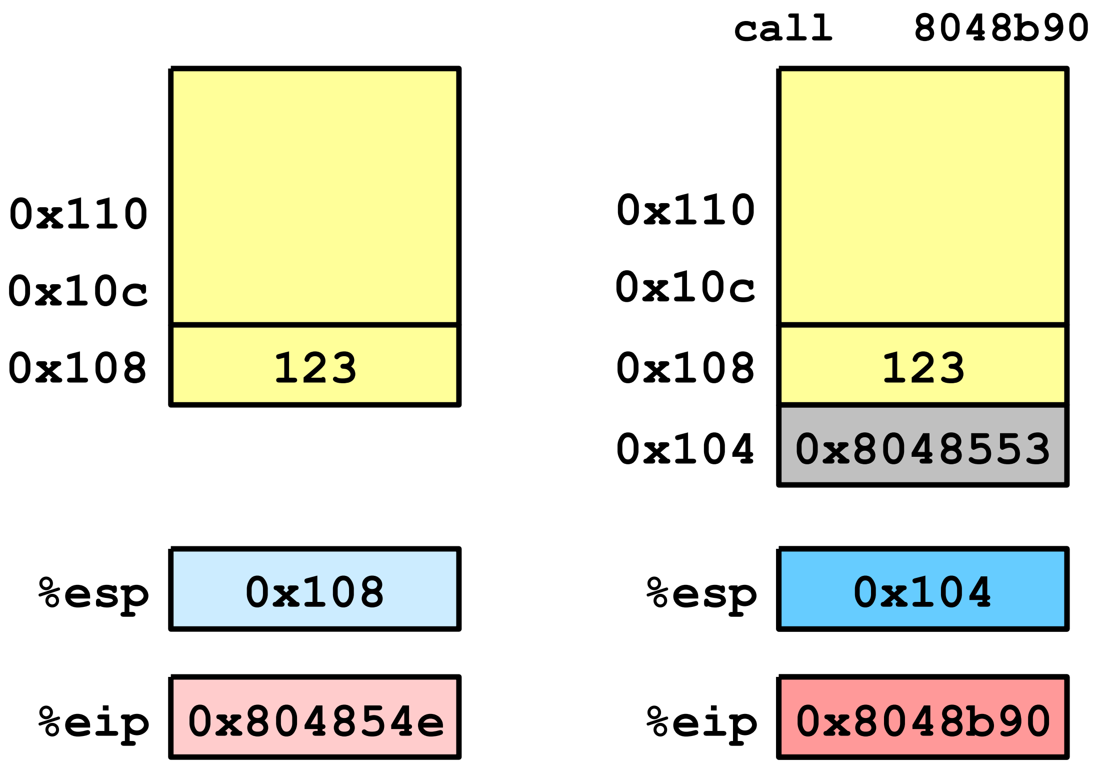
8.7 Program control

- Subroutine call and return (cont.)
  - Most efficient way is to store the return address in a memory stack
  - Subroutine call
    - SP ← SP + 1
    - M[SP] ← PC
    - PC ← EA
  - Return from subroutine
    - PC ← M[SP]
    - SP ← SP - 1
  - Memory stack is effective for recursive subroutine

8.7 Program control

- Subroutine call and return (cont.)

```
    804854e: e8 3d 06 00 00        call   8048b90 <main>
    8048553: 50                    pushl   %eax
```

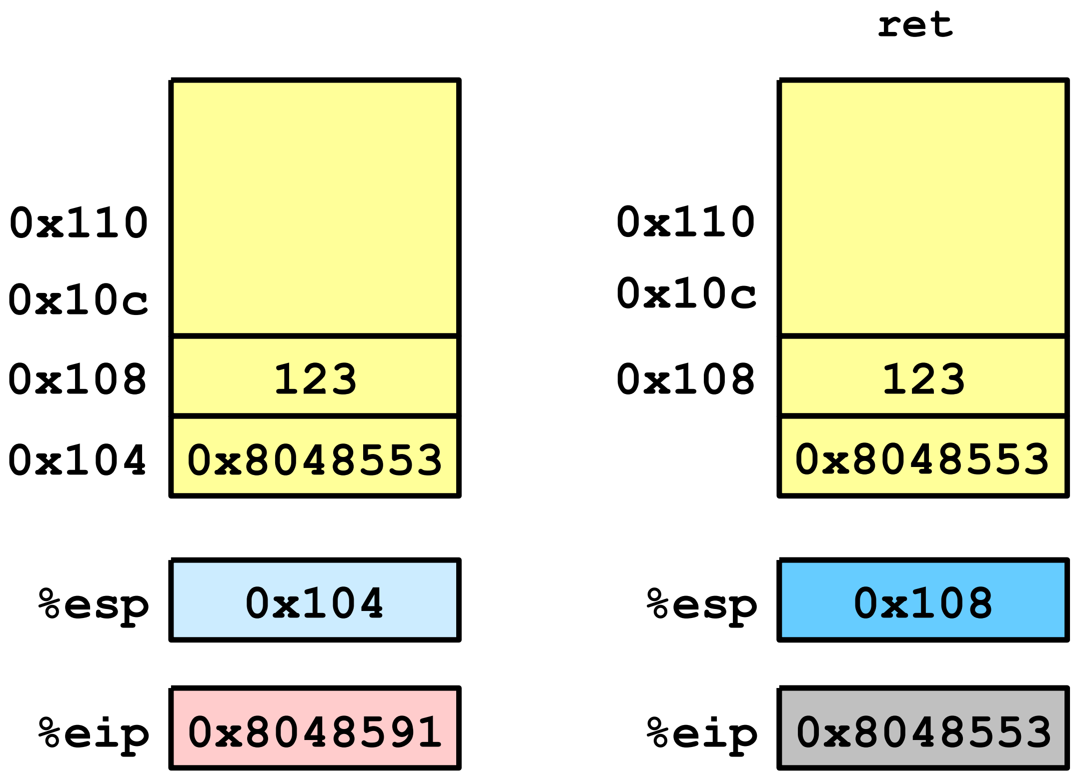


`%eip` is program counter

8.7 Program control

- Subroutine call and return (cont.)

```
    8048591:  c3                              ret
```



8.7 Program control

- Program interrupt
  - Differences from subroutine call
    - 1. interrupt is initiated by an internal or external signal
    - 2. the address of the interrupt service program is determined by hardware
    - 3. interrupt procedure store all the information necessary to define the state of the CPU
  - The state of the CPU is determined from
    - 1. the content of the program counter
    - 2. the content of all processor registers
    - 3. the content of certain status conditions

8.7 Program control

- Program interrupt (cont.)
  - Program status word (PSW)
    - Collection of all status bit conditions in the CPU
  - Types of interrupts
    - 1. external interrupt
      - Comes form I/O devices, timing devices, circuit monitoring power supply, etc.
    - 2. internal interrupts (traps)
      - Invalid operations, register overflow, division by zero, etc.
    - 3. software interrupts
      - Supervisor call

8.8 Reduced instruction set computer (RISC)

- CISC characteristics (Complex Instruction Set Computer)
  - Provide a single machine instruction for each statement
  - Ex) DEC VAX, IBM 370 (1970's), Intel 16-bit

  - A large number of instructions
  - A large variety of addressing modes
  - Variable length instruction formats
  - Instructions that manipulate operands in memory

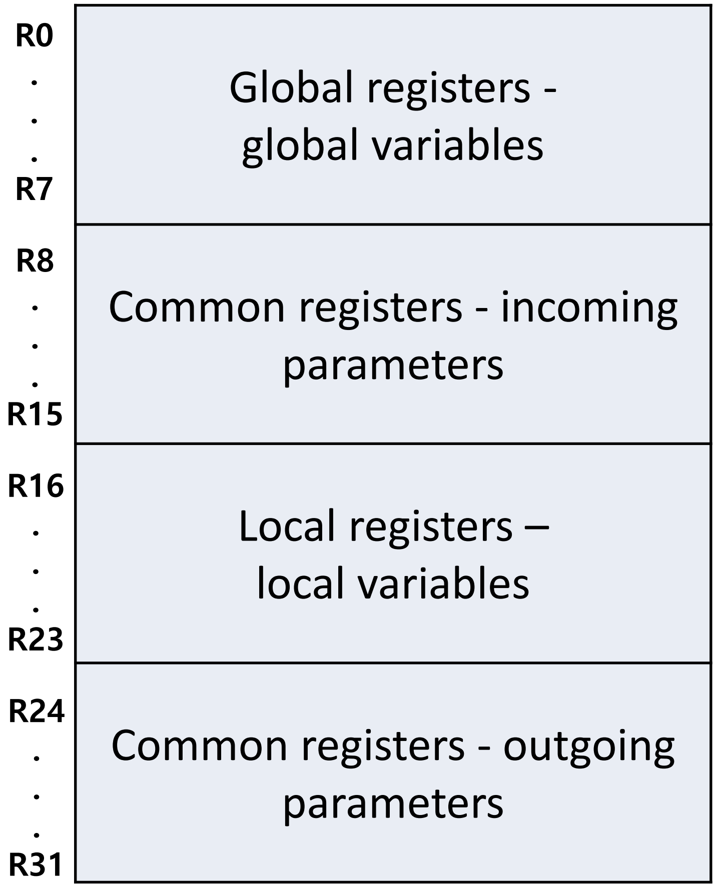
8.8 Reduced instruction set computer (RISC)

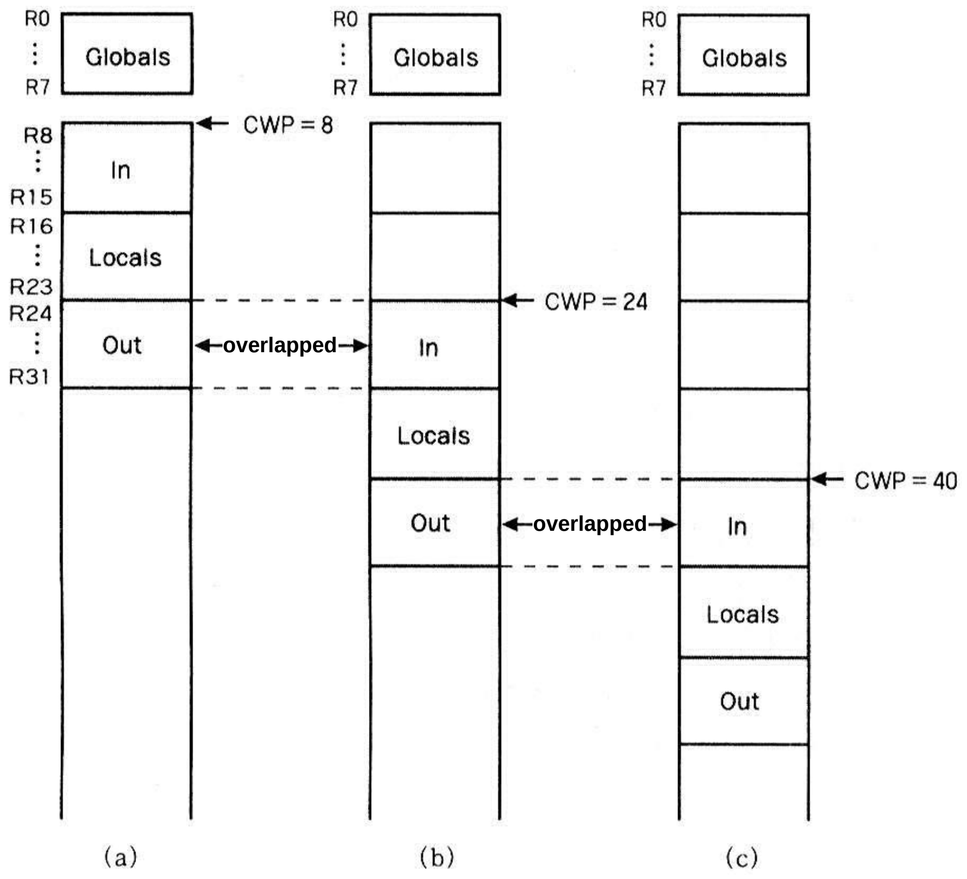
- RISC characteristics (Reduced Instruction Set Computer)
  - Relatively few instructions
  - Relatively few addressing modes
  - Memory access limited to load and store instructions
  - All operations done within the registers of the CPU
  - Fixed length easily decoded instruction format
  - Single cycle instruction execution
  - Hardwired control
  - Ex) IBM 801, RSIC I, MIPS (1980's), ARM's

  - A relatively large number of registers
    - Useful for storing intermediate results & for optimizing operand references
  - Use of overlapped register window
  - Efficient instruction pipeline
  - Compiler support for efficient translation

8.8 Reduced instruction set computer (RISC)

- Overlapped register windows
  - Global registers, local registers
  - Registers common to two windows


R0 . . . R7
Global registers - global variables
R8 . . . R15
Common registers - incoming parameters
R16 . . . R23
Local registers – local variables
R24 . . . R31
Common registers - outgoing parameters



CWP: current window pointer

8.8 Reduced instruction set computer (RISC)

- Overlapped register windows
  - Window size = L + 2C + G
    - = # of available registers for each window
  - Register file = (L + C)W + G
    - = # of all registers

  - G: # of global registers
  - L: # of local registers at each window
  - C: # of common registers over windows
  - W: # of windows

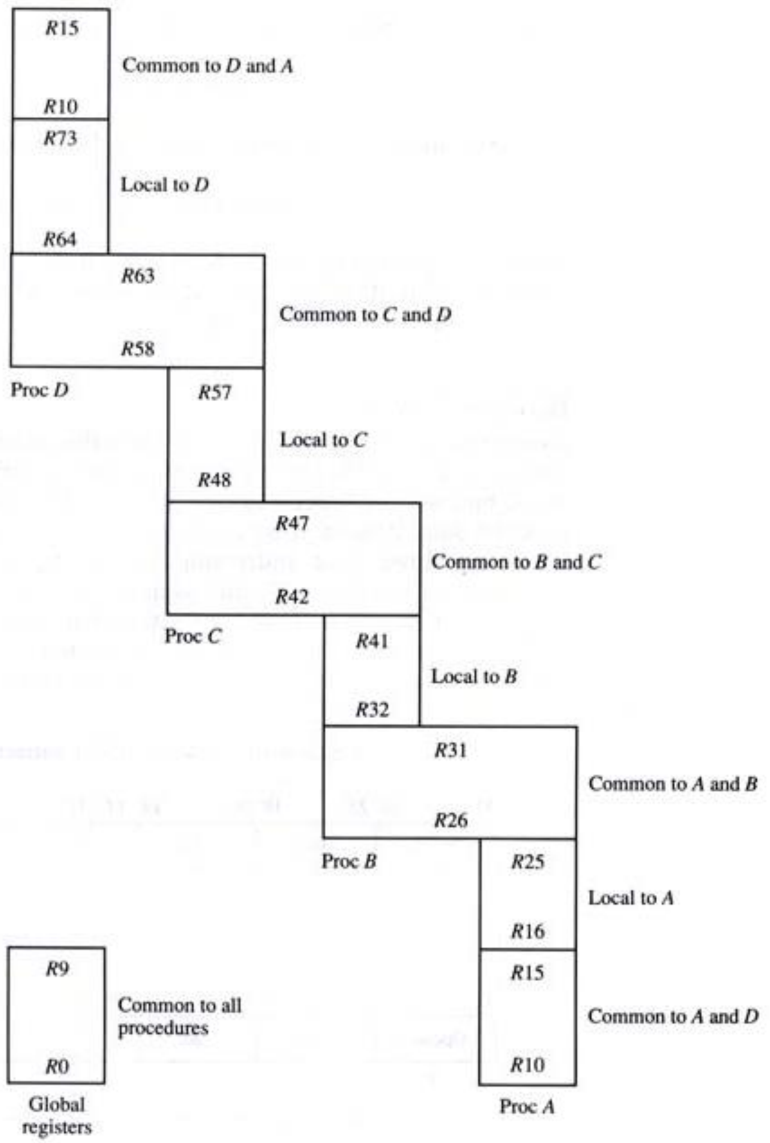
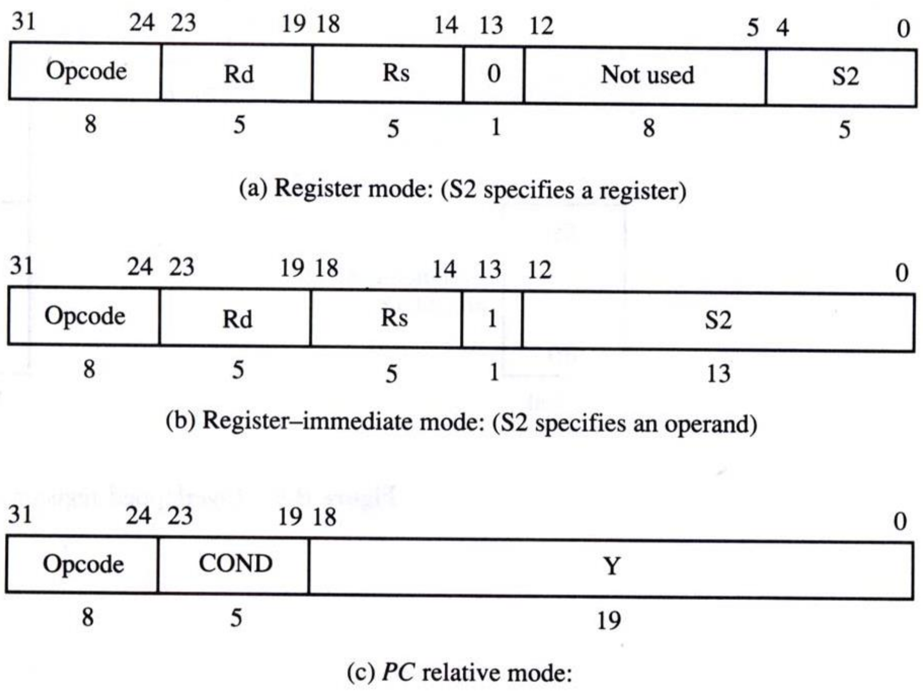


Figure 8-9 Overlapped register windows.

8.8 Reduced instruction set computer (RISC)

- Berkeley RISC I
  - 32-bit CPU
  - 32-bit address, 8/16/32-bit data
  - 31 instructions
  - Register, immediate, relative addressing modes
  - W: 8, G: 10, L: 10, C: 6
  - Register file: 138
    - (10 + 6) * 8 + 10
    - # of total registers
  - Window size: 32
    - 10 + 10 + 2*6
    - # of registers per window

Figure 8-10 Berkeley RISC I instruction formats.



31 24 23 19 18 14 13 12 5 4 0
Opcode | Rd | Rs | 0 | Not used | S2
8 5 5 1 8 5
(a) Register mode: (S2 specifies a register)
31 24 23 19 18 14 13 12 0
Opcode | Rd | Rs | 1 | S2
8 5 5 1 13
(b) Register–immediate mode: (S2 specifies an operand)
31 24 23 19 18 0
Opcode | COND | Y
8 5 19
(c) PC relative mode:

8.8 Reduced instruction set computer (RISC)

- Berkeley RISC I
  - Data manipulation (12)
  - Data transfer (11)
  - Program control (8)

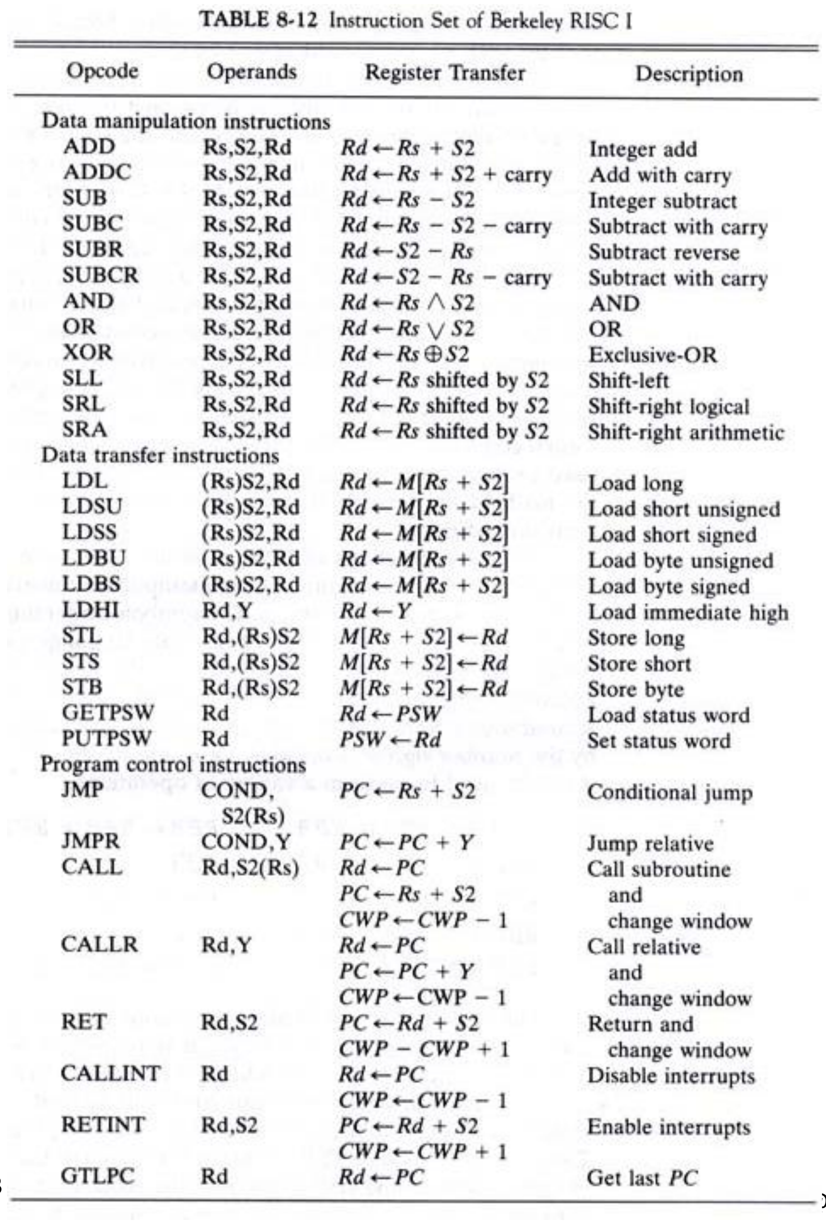
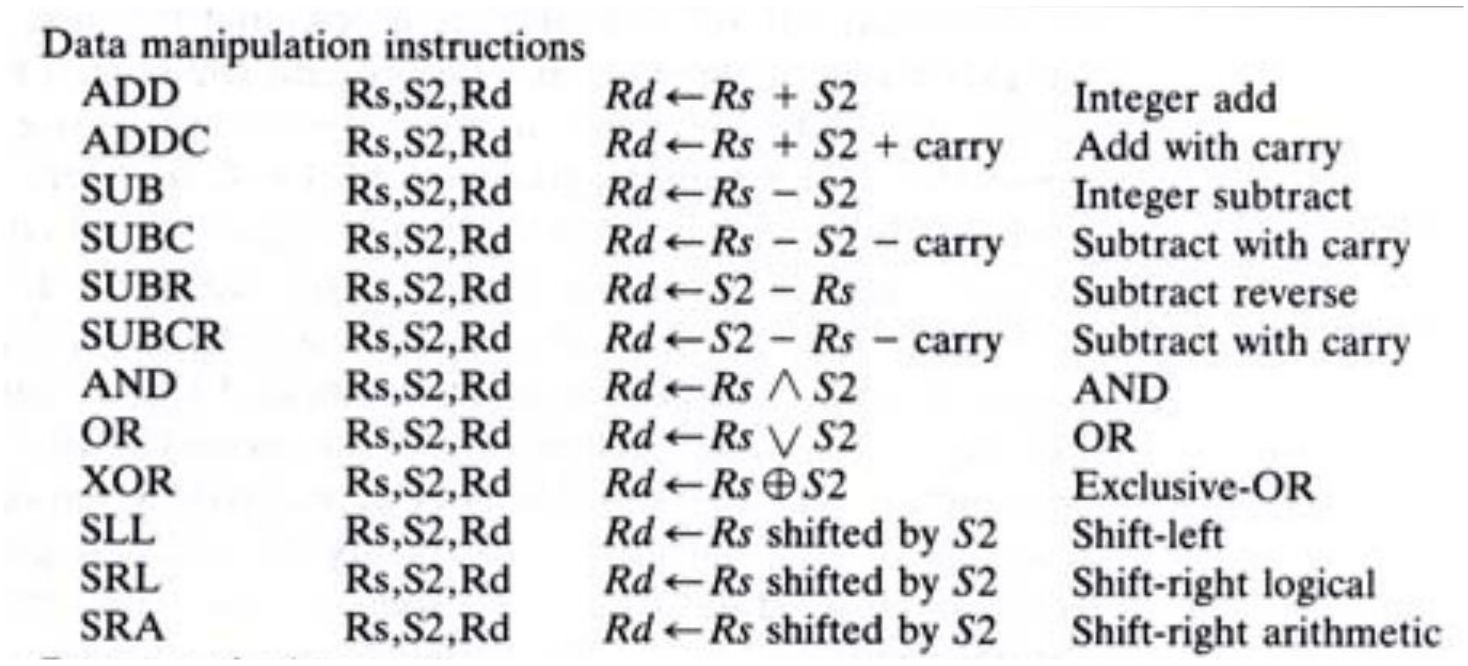


TABLE 8-12 Instruction Set of Berkeley RISC I

Opcode	Operands	Register Transfer	Description
Data manipulation instructions			
ADD	Rs,S2,Rd	$Rd \leftarrow Rs + S2$	Integer add
ADDC	Rs,S2,Rd	$Rd \leftarrow Rs + S2 + \text{carry}$	Add with carry
SUB	Rs,S2,Rd	$Rd \leftarrow Rs - S2$	Integer subtract
SUBC	Rs,S2,Rd	$Rd \leftarrow Rs - S2 - \text{carry}$	Subtract with carry
SUBR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs$	Subtract reverse
SUBCR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs - \text{carry}$	Subtract with carry
AND	Rs,S2,Rd	$Rd \leftarrow Rs \wedge S2$	AND
OR	Rs,S2,Rd	$Rd \leftarrow Rs \vee S2$	OR
XOR	Rs,S2,Rd	$Rd \leftarrow Rs \oplus S2$	Exclusive-OR
SLL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-left
SRL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-right logical
SRA	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-right arithmetic
Data transfer instructions			
LDL	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load long
LDSU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short unsigned
LDSS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short signed
LDBU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte unsigned
LDBS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte signed
LDHI	Rd,Y	$Rd \leftarrow Y$	Load immediate high
STL	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store long
STS	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store short
STB	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store byte
GETPSW	Rd	$Rd \leftarrow PSW$	Load status word
PUTPSW	Rd	$PSW \leftarrow Rd$	Set status word
Program control instructions			
JMP	COND, S2(Rs)	$PC \leftarrow Rs + S2$	Conditional jump
JMPR	COND,Y	$PC \leftarrow PC + Y$	Jump relative
CALL	Rd,S2(Rs)	$Rd \leftarrow PC$ $PC \leftarrow Rs + S2$ $CWP \leftarrow CWP - 1$	Call subroutine and change window
CALLR	Rd,Y	$Rd \leftarrow PC$ $PC \leftarrow PC + Y$ $CWP \leftarrow CWP - 1$	Call relative and change window
RET	Rd,S2	$PC \leftarrow Rd + S2$ $CWP - CWP + 1$	Return and change window
CALLINT	Rd	$Rd \leftarrow PC$ $CWP \leftarrow CWP - 1$	Disable interrupts
RETINT	Rd,S2	$PC \leftarrow Rd + S2$ $CWP \leftarrow CWP + 1$	Enable interrupts
GTLPC	Rd	$Rd \leftarrow PC$	Get last PC

8.8 Reduced instruction set computer (RISC)

- Berkeley RISC I
  - Data manipulation



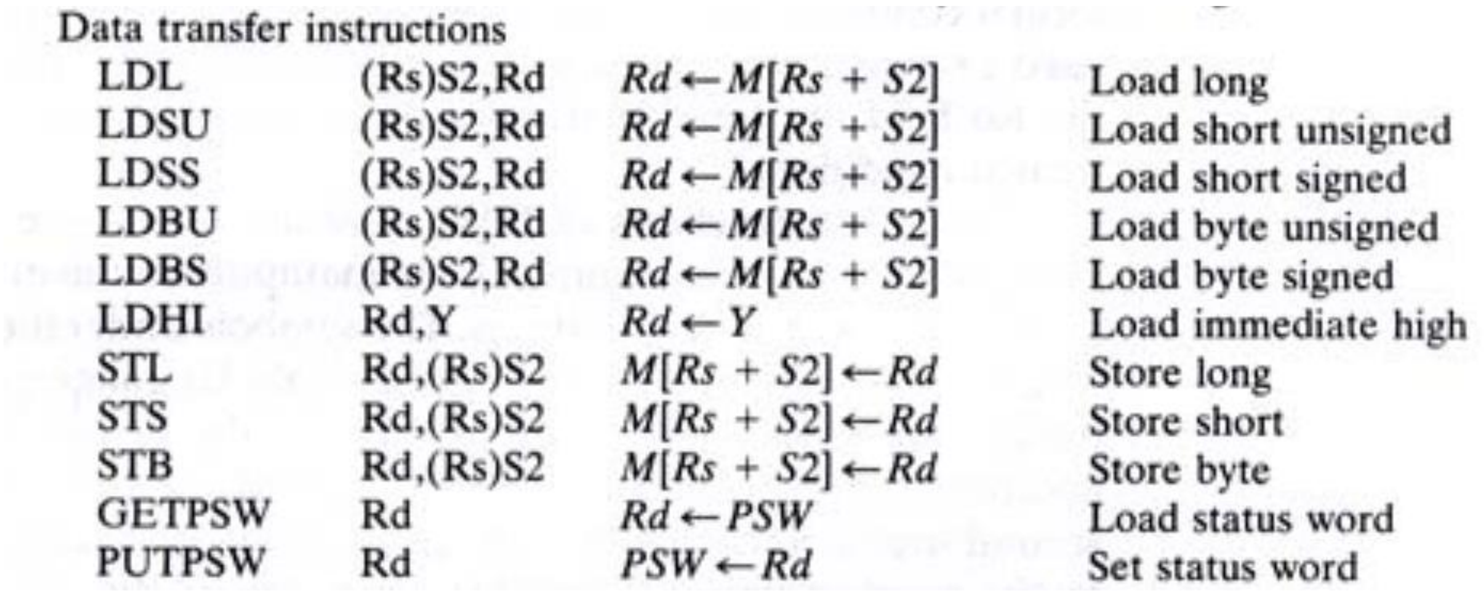
Data manipulation instructions

ADD	Rs,S2,Rd	$Rd \leftarrow Rs + S2$	Integer add
ADDC	Rs,S2,Rd	$Rd \leftarrow Rs + S2 + \text{carry}$	Add with carry
SUB	Rs,S2,Rd	$Rd \leftarrow Rs - S2$	Integer subtract
SUBC	Rs,S2,Rd	$Rd \leftarrow Rs - S2 - \text{carry}$	Subtract with carry
SUBR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs$	Subtract reverse
SUBCR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs - \text{carry}$	Subtract with carry
AND	Rs,S2,Rd	$Rd \leftarrow Rs \wedge S2$	AND
OR	Rs,S2,Rd	$Rd \leftarrow Rs \vee S2$	OR
XOR	Rs,S2,Rd	$Rd \leftarrow Rs \oplus S2$	Exclusive-OR
SLL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-left
SRL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-right logical
SRA	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by $S2$	Shift-right arithmetic

ADD R22, R21, R23	R23 $\leftarrow$ R22 + R21
ADD R22, #150, R23	R23 $\leftarrow$ R22 + 150
ADD R0, R21, R22	R22 $\leftarrow$ R21
ADD R0, #150, R22	R22 $\leftarrow$ 150
ADD R22, #1, R22	R22 $\leftarrow$ R22 + 1

8.8 Reduced instruction set computer (RISC)

- Berkeley RISC I
  - Data transfer



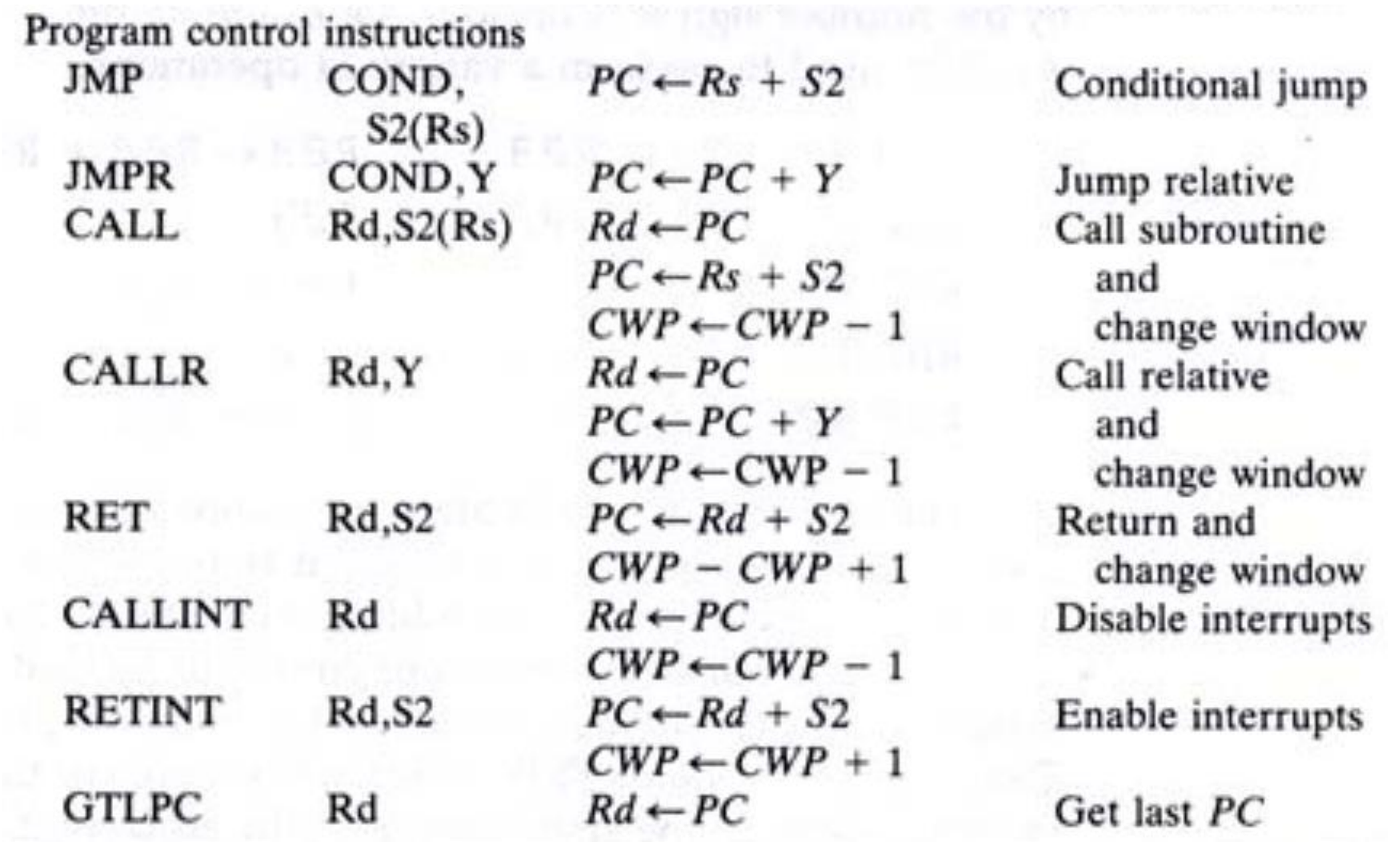
Data transfer instructions

LDL	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load long
LDSU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short unsigned
LDSS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load short signed
LDBU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte unsigned
LDBS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	Load byte signed
LDHI	Rd,Y	$Rd \leftarrow Y$	Load immediate high
STL	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store long
STS	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store short
STB	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	Store byte
GETPSW	Rd	$Rd \leftarrow PSW$	Load status word
PUTPSW	Rd	$PSW \leftarrow Rd$	Set status word

LDL (R22) #150, R5	R5 ← M[R22+150]
LDL (R22) #0, R5	R5 ← M[R22]
LDL (R0) #500, R5	R5 ← M[500]

8.8 Reduced instruction set computer (RISC)

- Berkeley RISC I
  - Program control
    - 3-bit CWP



Program control instructions

JMP	COND, S2(Rs)	$PC \leftarrow Rs + S2$	Conditional jump
JMPR	COND,Y	$PC \leftarrow PC + Y$	Jump relative
CALL	Rd,S2(Rs)	$Rd \leftarrow PC$	Call subroutine
		$PC \leftarrow Rs + S2$	and
		$CWP \leftarrow CWP - 1$	change window
CALLR	Rd,Y	$Rd \leftarrow PC$	Call relative
		$PC \leftarrow PC + Y$	and
		$CWP \leftarrow CWP - 1$	change window
RET	Rd,S2	$PC \leftarrow Rd + S2$	Return and
		$CWP - CWP + 1$	change window
CALLINT	Rd	$Rd \leftarrow PC$	Disable interrupts
		$CWP \leftarrow CWP - 1$	
RETINT	Rd,S2	$PC \leftarrow Rd + S2$	Enable interrupts
		$CWP \leftarrow CWP + 1$	
GTLPC	Rd	$Rd \leftarrow PC$	Get last *PC*

Problems

- 8-3, 8-4, 8-7, 8-8, 8-9, 8-14, 8-18,
- 8-25, 8-26, 8-27, 8-31, 8-32, 8-37