8-1. Introduction

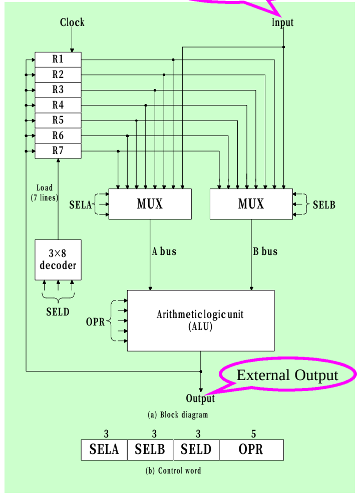
- 8-1 Introduction
  - 3 major parts of CPU : *Fig. 8-1*
    - 1) Register Set
    - 2) ALU
    - 3) Control
  - Design Examples of simple CPU
    - Hardwired Control : *Chap. 5*
    - Microprogrammed Control : *Chap. 7*
  - In this chapter : *Chap. 8* ← *Computer Architecture as seen by the programmer*
    - Describe the organization and architecture of the CPU with *an emphasis on the user's view of the computer*
    - User who programs the computer in machine/assembly language *must be aware* of
      - » 1) Instruction Formats
      - » 2) Addressing Modes
      - » 3) Register Sets

      } Chap. 8 의 주요 내용
    - The last section presents the concept of *Reduced Instruction Set Computer* (*RISC*)

8-2. General Register Organization

- 8-2 General Register Organization
  - Register의 필요성
    - Memory locations are needed for storing *pointers, counters, return address, temporary results, and partial products* during multiplication (*in the programming examples of Chap. 6*)
    - Memory access is the most time-consuming operation in a computer
    - More convenient and efficient way is to store intermediate values in processor registers
  - Bus organization for 7 CPU registers : *Fig. 8-2*
    - **2 MUX** : select one of 7 register or external data input by **SELA** and **SELB**
    - **BUS A and BUS B** : form the inputs to a common ALU
    - **ALU** : **OPR** determine the arithmetic or logic microoperation
      - » The result of the microoperation is available for external data output and also goes into the inputs of all the registers
    - **3 X 8 Decoder** : select the register (by **SELD**) that receives the information from ALU

External Input



External Output

8-2. General Register Organization

- Binary selector input : 예제 $R1 \leftarrow R2 + R3$
  - 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 the arithmetic addition R2 + R3
  - 4) Decoder selector (**SELD**) : to transfer the content of the output bus into R1
- Control Word
  - 14 bit control word (*4 fields*) : *Fig. 8-2(b)*
    - » **SELA** (*3 bits*) : select a source register for the A input of the ALU
    - » **SELB** (*3 bits*) : select a source register for the B input of the ALU
    - » **SELD** (*3 bits*) : select a destination register using the 3 X 8 decoder

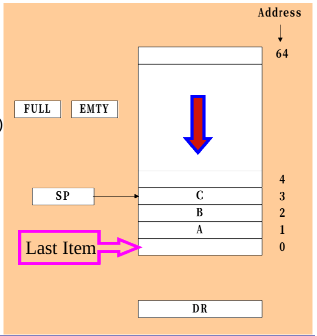
      } *Tab. 8-1*
    - » **OPR** (*5 bits*) : select one of the operations in the ALU → *Tab. 8-2*
  - Encoding of Register Selection Fields : *Tab. 8-1*
    - » **SELA** or **SELB** = 000 (**Input**) : MUX selects the external input data
    - » **SELD** = 000 (**None**) : no destination register is selected but the contents of the output bus are available in the external output
  - Encoding of ALU Operation (**OPR**) : *Tab. 8-2*

    Control Word를 Control Memory에 저장하여 Microprogrammed Control 방식으로 제어 가능함
- Examples of Microoperations : *Tab. 8-3*
  - TSFA (Transfer A) : $R7 \leftarrow R1$, $External\ Output \leftarrow R2$, $External\ Output \leftarrow External\ Input$
  - XOR : $R5 \leftarrow 0\ (XOR\ R5 \oplus R5)$

8-3. Stack Organization

- 8-3 Stack Organization
  - Stack or LIFO(*Last-In, First-Out*)
    - A storage device that stores information
      - » The item stored last is the first item retrieved = a stack of tray
    - Stack Pointer (**SP**)
      - » The register that holds the address for the stack
      - » SP always points at the top item in the stack
    - Two Operations of a stack : *Insertion and Deletion of Items*
      - » PUSH : Push-Down = Insertion
      - » POP : Pop-Up = Deletion
    - Stack의 종류
      - » 1) Register Stack (*Stack Depth가 제한*)
        - a finite number of memory words or register(*stand alone*)
      - » 2) Memory Stack (*Stack Depth가 유동적*)
        - a portion of a large memory
  - Register Stack : *Fig. 8-3*
    - PUSH :
      $SP \leftarrow SP + 1$: Increment SP
      $M[SP] \leftarrow DR$: Write to the stack
      $If\ (SP = 0)\ then\ (FULL \leftarrow 1)$: Check if stack is full
      $EMTY \leftarrow 0$: Mark not empty

* 초기 상태
SP = 0,
EMTY = 1,
FULL = 0



8-3. Stack Organization

- » The first item is stored at *address 1*, and the last item is stored at *address 0*
- POP : $DR \leftarrow M[SP]$: Read item from the top of stack
  $SP \leftarrow SP - 1$: Decrement Stack Pointer
  $If\ (SP = 0)\ then\ (EMTY \leftarrow 1)$: Check if stack is empty
  $FULL \leftarrow 0$: Mark not full

* Memory Stack
PUSH = Address 감소
* Register Stack
PUSH = Address 증가

◆ Memory Stack : ***Fig. 8-4***

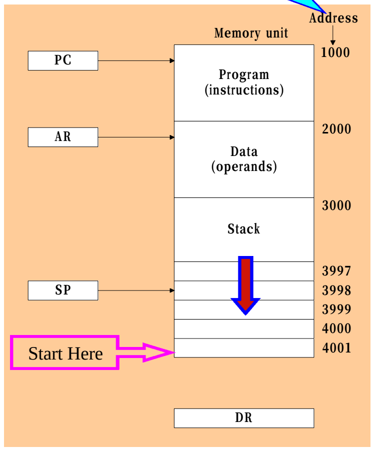
- PUSH : $SP \leftarrow SP - 1$
  $M[SP] \leftarrow DR$
  - » The first item is stored at *address 4000*
- POP : $DR \leftarrow M[SP]$
  $SP \leftarrow SP + 1$

* 초기 상태
SP = 4001

* Error Condition
PUSH when FULL = 1
POP when EMTY = 1

◆ Stack Limits

- Check for stack overflow(*full*)/underflow(*empty*)
  - » Checked by using two register
    - Upper Limit and Lower Limit Register
  - » After PUSH Operation
    - SP compared with the upper limit register
  - » After POP Operation
    - SP compared with the lower limit register

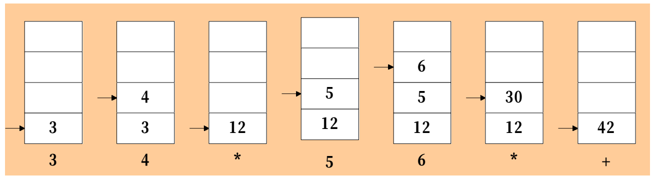


8-4. Instruction Formats

◆ RPN (*Reverse Polish Notation*)

Stack을 이용한 Arithmetic

- The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer
- A stack organization is very effective for evaluating arithmetic expressions
- 예제) **A * B + C * D** → **AB * CD * +** : ***Fig. 8-5***
  - » (3 * 4) + (5 * 6) → 34 * 56 * +



■ 8-4 Instruction Formats

◆ Fields in Instruction Formats

- 1) **Operation Code Field** : specify the operation to be performed
- 2) **Address Field** : designate a memory address or a processor register
- 3) **Mode Field** : specify the operand or the effective address (*Addressing Mode*)

8-4. Instruction Formats

◆ 3 types of CPU organizations

X = Operand Address

- 1) Single AC Org. : **ADD X** $AC \leftarrow AC + M[X]$
- 2) General Register Org. : **ADD R1, R2, R3** $R1 \leftarrow R2 + R3$
- 3) Stack Org. : **PUSH X** $TOS \leftarrow M[X]$

◆ The influence of the number of addresses on computer instruction

[예제] 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 operation to and from ***memory*** and ***AC register*** **: STORE, LOAD**
- Operand memory addresses **: A, B, C, D**
- Result memory address **: X**

- 1) Three-Address Instruction

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

- » Each address fields specify either a processor register or a memory operand
- » Short program
- » Require too many bit to specify 3 address

8-4. Instruction Formats

- 2) Two-Address Instruction

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	$M[X] \leftarrow R1$

- » The most common in commercial computers
- » Each address fields specify either a processor register or a memory operand

- 3) One-Address Instruction

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

- » All operations are done between the AC register and memory operand

8-4. Instruction Formats

- 4) Zero-Address Instruction

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	$M[X] \leftarrow TOS$

  - » 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 operand
  - » Zero-Address : absence of address (**ADD, MUL**)

◆ 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
- RISC architecture will be explained in *Sec. 8-8*

8-5. Addressing Modes

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

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

◆ Addressing Mode의 필요성
- 1) To give programming versatility to the user
  - » pointers to memory, counters for loop control, indexing of data, ….
- 2) To reduce the number of bits in the addressing field of the instruction

◆ Instruction Cycle
- 1) Fetch the instruction from memory and **PC + 1**
- 2) Decode the instruction
- 3) Execute the instruction

8-5. Addressing Modes

◆ Program Counter (***PC***)
- PC 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
- 1) Distinct Binary Code
  - » Instruction Format 에 Opcode 와 같이 별도에 Addressing Mode Field를 갖고 있음
- 2) Single Binary Code
  - » Instruction Format에 Opcode 와 Addressing Mode Field가 섞여 있음

◆ Instruction Format with mode field : *Fig. 8-6*

Opcode	Mode	Address

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

8-5. Addressing Modes

◆ Immediate Mode
- Operand field contains the actual operand
- Useful for initializing 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** $AC \leftarrow R1$ Implied Mode

◆ 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
  - » Register 를 select 하는 것이 bit 수가 적게 소요됨
- ***Example*** : **LD (R1)** $AC \leftarrow M[R1]$

◆ Autoincrement or Autodecrement Mode
- Similar to the 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

8-5. Addressing Modes

- ***Example*** (*Autoincrement*) : **LD (R1)+** $AC \leftarrow M[R1],\ R1 \leftarrow R1+1$

◆ Direct Addressing Mode

- Effective address is equal to the address field of the instruction (Operand)
- Address field specifies the actual branch address in a branch-type instruction
- ***Example*** : **LD ADR** $AC \leftarrow M[ADR]$ ADR = Address part of Instruction

◆ Indirect Addressing Mode

- Address field of instruction gives the address where the effective address is stored in memory
- ***Example*** : **LD @ADR** $AC \leftarrow M[M[ADR]]$

◆ Relative Addressing Mode

- PC is added to the address part of the instruction to obtain the effective address
- ***Example*** : **LD $ADR** $AC \leftarrow M[PC + ADR]$

◆ Indexed Addressing Mode

- XR (*Index register*) is added to the address part of the instruction to obtain the effective address
- ***Example*** : **LD ADR(XR)** $AC \leftarrow M[ADR + XR]$

◆ Base Register Addressing Mode Not Here

- the content of a base register is added to the address part of the instruction to obtain the effective address

8-5. Addressing Modes

- Similar to the indexed addressing mode except that the register is now called a *base register* instead of an *index register*
  - » index register (XR) : **LD ADR(XR)** $AC \leftarrow M[ADR + XR]$ ADR 기준
    - ▪ index register hold an index number that is relative to the address part of the instruction
  - » base register (BR) : **LD ADR(BR)** $AC \leftarrow M[BR + ADR]$ BR 기준
    - ▪ base register hold a base address
    - ▪ the address field of the instruction gives a displacement relative to this base address

◆ 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

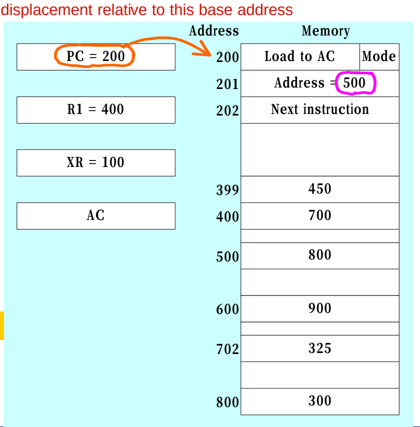
R1 = 400

500 + 202 (PC)

500 + 100 (XR)

R1 = 400 (after)

R1 = 400 -1 (prior)



8-6. Data Transfer and Manipulation

■ 8-6 Data Transfer and Manipulation

◆ Most computer instructions can be classified into three categories:

- 1) Data transfer, 2) Data manipulation, 3) Program control instructions

◆ Data Transfer Instruction

- Typical Data Transfer Instruction : ***Tab. 8-5***
  - » **Load** : transfer from memory to a processor register, usually an AC (*memory read*)
  - » **Store** : transfer from a processor register into memory (*memory write*)
  - » **Move** : transfer from one register to another register
  - » **Exchange** : swap information between two registers or a register and a memory word
  - » **Input/Output** : transfer data among processor registers and input/output device
  - » **Push/Pop** : transfer data between processor registers and a memory stack
- 8 Addressing Mode for the LOAD Instruction : Tab. 8-6
  - » **@** : Indirect Address
  - » **$** : Address relative to PC
  - » **#** : Immediate Mode
  - » **( )** : Index Mode, Register Indirect, Autoincrement 에서 register 지정

◆ Data Manipulation Instruction

- 1) Arithmetic, 2) Logical and bit manipulation, 3) Shift Instruction

8-7. Program Control

- Arithmetic Instructions : ***Tab. 8-7***
- Logical and Bit Manipulation Instructions : ***Tab. 8-8***
- Shift Instructions : ***Tab. 8-9***

■ 8-7 Program Control

◆ Program Control Instruction : ***Tab. 8-10***

- Branch and Jump instructions are used interchangeably to mean the same thing

◆ Status Bit Conditions : ***Fig. 8-8***

- Condition Code Bit or Flag Bit
  - » The bits are set or cleared as a result of an operation performed in the ALU

◆ 4-bit status register

- Bit **C** (*carry*) : set to 1 if the end carry C_8 is 1
- Bit **S** (*sign*) : set to 1 if F_7 is 1
- Bit **Z** (*zero*) : set to 1 if the output of the ALU contains all 0's
- Bit **V** (*overflow*) : set to 1 if the exclusive-OR of the last two carries (C_8 and C_7) is equal to 1
- ***Flag Example*** : A - B = A + (2's Comp. Of B) : ***A =11110000, B = 00010100***

```
  11110000
+ 11101100 (2's comp. of B)
1 11011100
```

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

8-7. Program Control

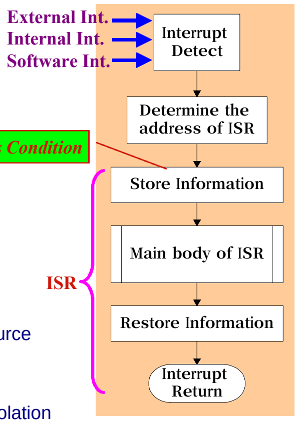
- Conditional Branch : *Tab. 8-11*
- Subroutine Call and Return
  - CALL : $SP \leftarrow SP - 1$: Decrement stack point
    $M[SP] \leftarrow PC$: Push content of PC onto the stack
    $PC \leftarrow Effective\ \ Address$: Transfer control to the subroutine
  - RETURN : $PC \leftarrow M[SP]$: Pop stack and transfer to PC
    $SP \leftarrow SP + 1$: Increment stack pointer
- Program Interrupt
  - Program Interrupt
    - » Transfer program control from a currently running program to another service 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 과 Subroutine Call 의 차이점
    - » 1) An interrupt is initiated by an internal or external signal (*except for software interrupt*)
      - A subroutine call is initiated from the execution of an instruction (CALL)
    - » 2) 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
    - » 3) An interrupt procedure stores all the information necessary to define the state of the CPU
      - A subroutine call stores only the program counter (*Return address*)

8-7. Program Control

- Program Status Word (PSW)
  - » The collection of all status bit conditions in the CPU
- Two CPU Operating Modes
  - » Supervisor (*System*) Mode : Privileged Instruction 실행
    - When the CPU is executing a program that is part of the operating system
  - » User Mode : User program 실행

PC, CPU Register, Status Condition

CPU operating mode is determined from special bits in the PSW



- Types of Interrupts
  - 1) External Interrupts
    - » come from I/O device, from a timing device, from a circuit monitoring the power supply, or from any other external source
  - 2) Internal Interrupts or TRAP
    - » caused by register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation
  - 3) Software Interrupts
    - » initiated by executing an instruction (**INT** or **RST**)
    - » used by the programmer to initiate an interrupt procedure at any desired point in the program

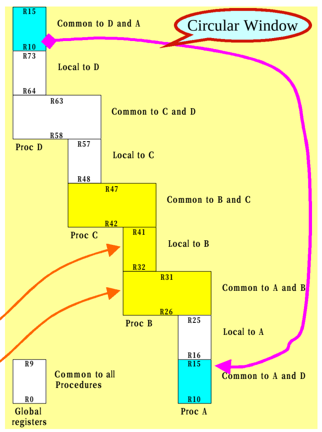
8-8. Reduced Instruction Set Computer (RISC)

- 8-8 Reduced Instruction Set Computer (RISC)
  - Complex Instruction Set Computer (CISC)
    - Major characteristics of a CISC architecture
      - » 1) A large number of instructions - typically from 100 to 250 instruction
      - » 2) Some instructions that perform specialized tasks and are used infrequently
      - » 3) A large variety of addressing modes - typically from 5 to 20 different modes
      - » 4) Variable-length instruction formats
      - » 5) Instructions that manipulate operands in memory (RISC 에서는 in register)
  - Reduced Instruction Set Computer (RISC)
    - Major characteristics of a RISC architecture
      - » 1) Relatively few instructions
      - » 2) Relatively few addressing modes
      - » 3) Memory access limited to load and store instruction
      - » 4) All operations done within the registers of the CPU
      - » 5) Fixed-length, easily decoded instruction format
      - » 6) Single-cycle instruction execution
      - » 7) Hardwired rather than microprogrammed control

8-8. Reduced Instruction Set Computer (RISC)

- Other characteristics of a RISC architecture
  - » 1) A relatively large number of registers in the processor unit
  - » 2) Use of overlapped register windows to speed-up procedure call and return
  - » 3) Efficient instruction pipeline
  - » 4) Compiler support for efficient translation of high-level language programs into machine language programs
- Overlapped Register Windows
  - Time consuming operations during procedure call
    - » Saving and restoring registers
    - » Passing of parameters and results
  - Overlapped Register Windows
    - » Provide the passing of parameters and avoid the need for saving and restoring register values ***by hardware***
- Concept of overlapped register windows : *Fig. 8-9*
  - Total 74 registers : R0 - R73
    - » R0 - R9 : Global registers
    - » R10 - R63 : 4 windows
      - Window A
      - Window B
      - Window C
      - Window D

10 Local registers
+
2 sets of 6 registers
(common to adjacent windows)



8-8. Reduced Instruction Set Computer (RISC)

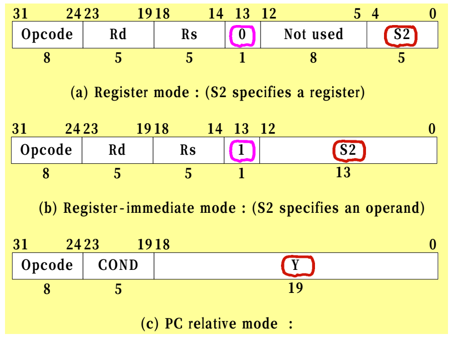
- ***Example)*** Procedure A calls procedure B
  - » R26 - R31
    - Store parameters for procedure B
    - Store results of procedure B
  - » R16 - R25 : Local to procedure A
  - » R32 - R41 : Local to procedure B
- Window Size = L + 2C + G = 10 + (2 X 6) + 10 = 32 registers
- Register File (total register) = (L + C) X W + G = (10 + 6) X 4 + 10 = 74 registers
  - » 여기서, **G** : Global registers = *10*
    - **L** : Local registers = *10*
    - **C** : Common registers = *6*
    - **W** : Number of windows = *4*

◆ Berkeley RISC I

- RISC Architecture 의 기원 : 1980 년대 초
  - » Berkeley RISC project : first project = **Berkeley RISC I**
  - » Stanford MIPS project
- Berkeley RISC I
  - » 32 bit CPU, 32 bit instruction format, 31 instruction
  - » 3 addressing modes : register, immediate, relative to PC

8-8. Reduced Instruction Set Computer (RISC)

- Instruction Set : ***Tab. 8-12***
- Instruction Format : ***Fig. 8-10***
- Register Mode : ***bit 13 = 0***
  - » S2 = register
  - » ***Example)*** **ADD R22, R21, R23**
    - **ADD Rs, S2, Rd :** Rd = Rs + S2
- Register Immediate Mode : ***bit 13 = 1***
  - » S2 = sign extended 13 bit constant
  - » ***Example)*** **LDL (R22)#150, R5**
    - **LDL (Rs)S2, Rd :** Rd = M[R22] + 150
- PC Relative Mode
  - » Y = 19 bit relative address
  - » ***Example)*** **JMPR COND, Y**
    - Jump to PC = PC + Y
  - » CWP (Current Window Pointer)
    - CALL, RET시 stack pointer 같이 사용됨
- RISC Architecture Originator



Architecture	Originator	Licensees
Alpha	DEC	Mitsubishi, Samsung
MIPS	MIPS Technologies	NEC, Toshiba
PA-RISC	Hewlett Packard	Hitachi, Samsung
PowerPC	Apple, IBM, Motorola	Bul
Sparc	Sun	Fujitsu, Hyundai
i960	Intel	Intel only (Embedded Controller)