Computer Organization

COMP2120

Qi Zhao

April 5, 2024

Instruction Set and Addressing II

Addressing Modes

- **Addressing Modes**: describe the way to calculate the (effective) address of an operand.
  built into the instruction set of a machine (no need to use extra instructions)
- Using addressing mode can reduce the size of program code, because no code is required for address calculation if the addressing mode is supported.
- However, the hardware will be more complicated.
- Modern processors provide a large number of registers. Hence the need to reference memory is reduced.
- According to statistics, only a few addressing mode is widely used.

Addressing Modes

- A = contents of an address field in the instruction
- R = contents of an address field in the instruction that refers to a register
- EA = actual (effective) address of the location containing the referenced operand
- (X) = contents of memory location X or register X
- When we say `R9=1`, it means that the content of register `R9` is 1

Immediate Addressing

- the instruction provides the **actual** value of the operand.

$$\text{Operand} = \text{A}$$

- Examples

```
MOV #200, R6 ; Move the value 200 to R6, R6=200
ADD R2,#4,R2 ; add the value 4 to R2, R4=R4+4
```

- There are two types of immediate addressing
  1. the immediate operand is small, and can be fitted into the instruction word itself (also called literal mode)

MOV	200	R6

  — e.g. The VAX-11 provides a 6-bit literal mode
  — MIPS provides a 16-bit immediate operand.

Immediate Addressing

2. The immediate operand is on the word following the current instruction,

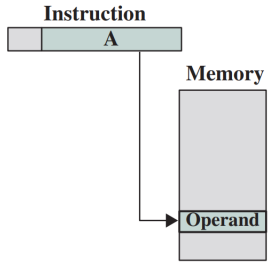
instruction
Immediate operand
Next instruction

- Format for both mode in assembly language programs are the same.
  For assembly program writing, no need to worry which mode to use, because the assembler will handle this.

Direct Addressing (Absolute Addressing)

- The operand field contain the address of the operand. It is the **address**, instead of the value, that is provided.

$$\mathrm{EA} = \mathrm{A}$$



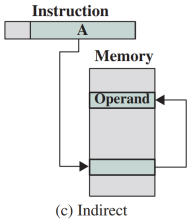
(b) Direct

- e.g. `MOV A, R1` (Move the content at address A to $R1$, $R1 = A$)
- common in earlier generations of computers
- provides only a limited address space

Indirect Addressing

- The data read is the address of the actual data. Usually indicated by a pair of parenthesis, e.g. (A).

$$\mathrm{EA} = (\mathrm{A})$$



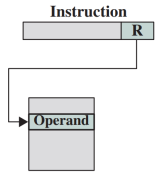
(c) Indirect

- has to read the memory once more to get the actual operand.
- e.g. `MOV (A), R1` (Move the content pointed by the value at address A to R1, `R1=mem[A]`)

Register Addressing

- the operand is in registers.

$$EA = R$$



(d) Register

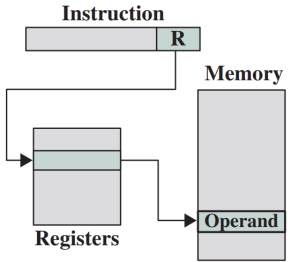
e.g. `MOV R1, R2` (move the content of R1 to R2, R2=R1)

- an address field for references registers will have from 3 to 5 bits, 8 to 32 general registers.
- It is the main addressing modes used in modern processors because of the speed of register access.

Register Indirect Addressing

- the address of the operand is in the specified register (pointer like). (R1), or @R1.

$$\mathrm{EA} = (\mathrm{R})$$



(e) Register indirect

Register Indirect Addressing

Example:

```
MOV (R1), R2; Move the content of mem[R1] to R2, R2=mem[R1]
```

	Memory		Register	
	Address	Content	Name	Content
Before	001000	55		
	001001	02	R1	00001000
	001002	3F	R2	1043834F
	001003	00		
After	001000	55		
	001001	02	R1	00001000
	001002	3F	R2	55023F00
	001003	00		

Register Indirect Addressing

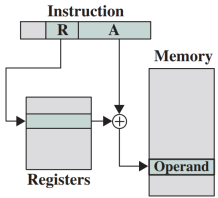
Example: (an assembly lang prog to calculate the sum of an array), $\sum_{i=0}^{1023} A[i]$
Assume a 3-operand instruction set:

```
        XOR R1,R1,R1    ;init R1 to 0, R1=0
        MOV #A,R2       ;R2=addr of A[0]
        MOV #1024,R3    ;R3 =size of array, 1024
LOOP:   ADD R1,(R2),R1  ;R1=R1+mem[R2]
                         mem[R2]=A[0]
        ADD R2,#4,R2    ;incr R2 to next elt, R2=R2+4
        SUB R3,#1,R3    ;decrement R3
        BNE LOOP        ;if R3!=0 If not, go back to Loop.
```

Displacement Addressing

- address of memory is given by: register + an offset.

$$EA = A + (R)$$



(f) Displacement

- Register: PC, Base pointer register
- Used to access local variables and parameters in function calls.
- Can also be used for accessing arrays. For example

```
INC 10(R5) ; increase the contents of R5+10 by one
```

Displacement Addressing

Example:

```
MOV 10(R1), R2; Move the content of 10+ R1 to R2
```

	Memory		Register	
	Address	Content	Name	Content
Before	001010	55		
	001011	02	R1	00001000
	001012	3F	R2	1043834F
	001013	00		
After	001010	55		
	001011	02	R1	00001000
	001012	3F	R2	55023F00
	001013	00		

Example: Displacement Addressing

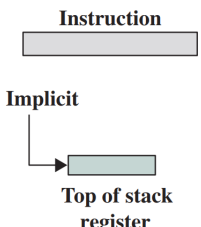
- $A[i] = A[i] + B[i], i = 1, 2 \ldots, 1024$ (here $+$ is OR operation)

```
        XOR R2,R2,R2   ; init R2=0
        MOV #1024, R3  ; init R3=size
LOOP:   MOV A(R2), R4  ; A=start addr of A[]
        MOV B(R2), R5  ; B=start addr of B[]
        OR R4, R5, R6  ; R4 OR R5
        ST R6, A(R2)
        ADD R2,#4,R2   ; increase R2 by 4
        SUB R3,#1,R3
        BNZ LOOP
```

Stack Addressing

- Some machine provide instructions such as `PUSH` and `POP` to access the stack.

$$EA = \text{top of stack}$$



(g) Stack

- operations for `PUSH` and `POP`
- stack pointer (SP) is a small register
- `PUSH R4:`

```
SUB SP, #4, SP
MOV R4,(SP)
```

- `POP R3:`

```
MOV (SP), R3
ADD SP, #4, SP
```

Summary

Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability