Computer Organization

COMP2120

Qi Zhao

March 24, 2024

Instruction Set and Addressing I

Elements of a Machine Instruction

- **Operation code, opcode**
  specifies the operation to be performed (ADD, LOAD)
- **Source operand reference** operands that are inputs for the operation, may involve one or more source operands.
- **Result operand reference** or destination operand. The operation may produce a result.
- **Next instruction reference** this tells the processor where to fetch the next instruction (except for Branch and procedure/function call)
- Each operation has up to 3 operands, 2 source operands and 1 destination operand.

Op code	src operand 1	src operand 2	dst operand
1 byte=8 bits	1 byte	1 byte	1 byte

Types of Operations

- Opcodes are represented by Mnemonics, or abbreviation of the instruction, e.g. `ADD` for add, `MUL` for multiply, `LD` for load etc.
- Abbreviation may be different for different machines. e.g. some machine uses `CALL` for function, while others may use `JSR` (Jump Subroutine).
- Usually, one can guess the meaning of the mnemonics fairly easily.
- Instruction can be classified into the following categories:
  - Data transfer — e.g. `MOV` (move), `LD` (load), and `ST` (store)
  - Arithmetic — e.g. `ADD`, `SUB`
  - Logical — e.g. `AND`, `OR`, and `NOT`
  - Transfer of Control — e.g. `BR` for branch, `CALL`
  - Input/Output — Many processors do not have Input/Output instructions. Instead, those I/O registers are placed in a normal memory map. (Memory-mapped I/O)
  - Data conversion — e.g. NEG, SXT (Sign Extend)

Example Instruction Operations

Table 12.3 Common Instruction Set Operations

Type	Operation Name	Description
Data transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT	(complement) Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Example Instruction Operations

Transfer of control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Example Instruction Operations

Table 12.4 Processor Actions for Various Types of Operations

Data transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write
Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags
Logical	Same as arithmetic
Conversion	Similar to arithmetic and logical. May involve special logic to perform conversion
Transfer of control	Update program counter. For subroutine call/return, manage parameter passing and linkage
I/O	Issue command to I/O module
	If memory-mapped I/O, determine memory-mapped address

Arithmetic vs Logical Operations

- Arithmetic operation – treat the operands as numbers, and has to consider the sign of the operands, in particular, on shift operations. (in 2's complement).
  - Shift operations are equivalent to arithmetic operations.
  - Shift Left — Multiply by 2.
  - Shift Right — Divide by 2 (and remainder is shifted out).
  - Shift operations has to perserve the sign of the operands if it is arithmetic.

  Shift operations are equivalent to arithmetic operations. Shift Left — Multiply by 2. Shift Right — Divide by 2 (and remainder is shifted out). Shift operations has to perserve the sign of the operands if it is arithmetic.
- Logical Operation — treat the operands as bit patterns.
- e.g. Arithmetic Shift Right has to preserve the sign, hence has to copy the original sign bit. (Not for logical shift operation)

Arithmetic vs Logical Operations

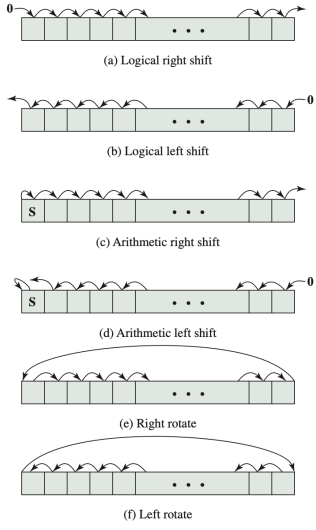


Figure 12.6 Shift and Rotate Operations

Arithmetic vs Logical Operations

Table 12.7 Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Transfer of Control

- loop: execute each instruction more than once and perhaps many thousands of times.
- decision making
- compose correctly a large or even medium-size computer program into small pieces
- Branch, skip and procedure call/return

Branch Instruction

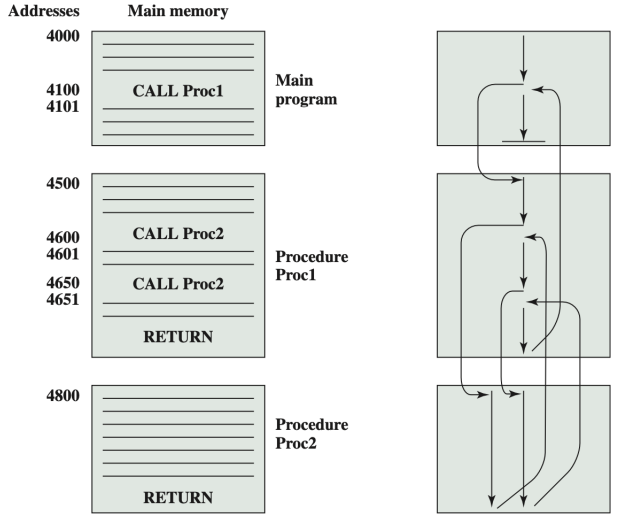
- Need to change the program flow based on some condition (if-else).
- also called a jump instruction, one of the operands indicts the address of the next instruction to be excecuted.
- The system has a one-bit flag for each condition, e.g. zero flag: if result is zero, then branch; result is non-zero, nothing happens.

Instruction	Meaning
`BR`	always goto (unconditional)
`BZ`	branch if zero flag is true
`BNZ`	branch if zero flag is not true

Procedure/Function call/return

- A **procedure** is a self- contained computer program that is incorporated into a larger program.
- At any point in the program the procedure may be invoked, or called.
- economy and modularity
- One procedure has
  a call instruction: branches from the present location to the procedure
  a return instruction: returns from the procedure to the place from which it was called.
- A special type of branch instruction where we need to store the address of next instruction to be executed (return address), so that on function return, the program can continue execution at the right place.
- Nested call requires the return addresses to be stored in some dynamic data structure.

Procedure/Function call/return



(a) Calls and returns

(b) Execution sequence

Procedure/Function call/return

- A procedure can be called from more than one location.
- A procedure call can appear in a procedure. This allows the nesting of procedures to an arbitrary depth.
- Each procedure call is matched by a return in the called program.

Procedure/Function call/return

How to store the return address

- Register
- Start of called procedure
* Top of stack
- The return addresses are retrieved in a Last-in-First-Out fashion (The last function called will be the first to return).
- Hence, we will store the return address in the system stack.

Procedure/Function call/return

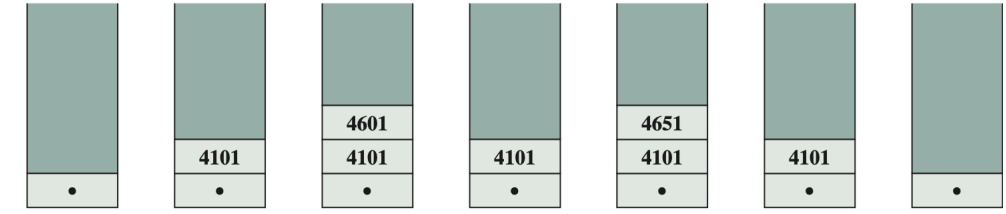


Figure 12.9 Use of Stack to Implement Nested Subroutines of Figure 12.8

Operands

- Number of operands for most instruction is always three — two source (src) and one destination (dst) operand.
- You may explicitly specify from 0 to 3 operands, the other operands are implied
  - 3 operand instruction — used in most instruction set nowadays (the one used in asg 2). e.g. `ADD A,B,C;` `C` ⟵ `A+B`
  - 2 operand instruction — one of the src operand also acts as the dst operand. e.g. `ADD A,B;` `B` ⟵ `A+B`
  - 1 operand instruction — a default operand which act as both src and dst (usually called the accumulator AC, hence called accumulator-based machine) e.g. `ADD A;` `AC` ⟵ `AC+A`
  - 0 operand instruction — all operands are implied. They are usually on the stack, i.e. Stack machine. e.g. `ADD` — Pop top 2 elements from stack and ADD, then push it back.

Operands

Table 12.1 Utilization of Instruction Addresses (Nonbranching Instructions)

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = accumulator
T = top of stack
(T − 1) = second element of stack
A, B, C = memory or register locations

Operands

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

Figure 12.3 Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Registers

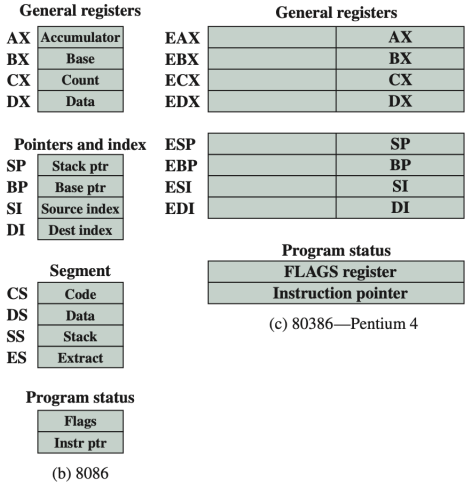
- The operands can either be in Memory or in Register (inside CPU)
- Two types of registers: general purpose registers vs dedicated purpose registers
- general purpose registers — registers can be used freely.
- Some dedicated purpose registers, such as Program Counter (PC) and Stack Pointer (SP).
- e.g. The 80x86 architecture defines a dedicated purpose registers, such as SP, and BP (base pointer for function calls)
  The VAX architecture (one type of complex instruction set computer (CISC)) where you can use any registers as a stack pointer (even for function call)
- There is a special register called PSW (processor status word) or flag register which define some condition codes

Example Flags in the PSW – x86

Table 12.8 x86 Status Flags

Status Bit	Name	Description
C	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
P	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
A	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
Z	Zero	Indicates that the result of an arithmetic or logic operation is 0.
S	Sign	Indicates the sign of the result of an arithmetic or logic operation.
O	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

Pentium Register File



Data Types

- two categories:
  - Numeric
    integers — signed and unsigned integers.
    Floating point — 32, 64 up to 128 bits. The MIPS provides special floating-point registers. (64-bit for double precision)
  - Non-numeric
    characters: either ASCII or unicode.
    Binary data (bit-map)
- Typical lengths are 8, 16, 32, and 64 bits.

MIPS Data Types

MIPS architectures: a family of reduced instruction set computer (RISC)
9 Basic Data Types:

- signed and unsigned bytes
- signed and unsigned halfwords
- signed and unsigned words
- doublewords (64 bits)
- single-precision floating point (32 bits)
- double-precision floating point (64 bits).