Chapter 2: Machines, Machine Languages, and Digital Logic

Topics

2.1 Classification of Computers and Their Instructions
2.2 Computer Instruction Sets
2.3 Informal Description of the Simple RISC Computer, SRC
2.4 Formal Description of SRC Using Register Transfer Notation, RTN
2.5 Describing Addressing Modes with RTN
2.6 Register Transfers and Logic Circuits: From Behavior to Hardware
What Must an Instruction Specify?

- Which operation to perform
  - Ans: Op code: add, load, branch, etc.
- Where to find the operand or operands
  - In CPU registers, memory cells, I/O locations, or part of instruction
- Place to store result
  - Again CPU register or memory cell
- Location of next instruction
  - Almost always memory cell pointed to by program counter—PC
  - Sometimes there is no operand, or no result, or no next instruction. Can you think of examples?

Data Flow

add r0, r1, r3

br endloop
Instructions Can Be Divided into 3 Classes

- Data movement instructions
  - Move data from a memory location or register to another memory location or register without changing its form
  - **Load**—source is memory and destination is register
  - **Store**—source is register and destination is memory
- Arithmetic and logic (ALU) instructions
  - Change the form of one or more operands to produce a result stored in another location
  - **Add, Sub, Shift**, etc.
- Branch instructions (control flow instructions)
  - Alter the normal flow of control from executing the next instruction in sequence
  - **Br Loc, Brz Loc2**,—unconditional or conditional branches
## Tbl 2.1 Examples of Data Movement Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, B</td>
<td>Move 16 bits from memory location A to Location B</td>
<td>VAX11</td>
</tr>
<tr>
<td>LDA A, Addr</td>
<td>Load accumulator A with the byte at memory location Addr</td>
<td>M6800</td>
</tr>
<tr>
<td>lwz R3, A</td>
<td>Move 32-bit data from memory location A to register R3</td>
<td>PPC601</td>
</tr>
<tr>
<td>li $3, 455</td>
<td>Load the 32-bit integer 455 into register $3</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>mov R4, dout</td>
<td>Move 16-bit data from R4 to output port dout</td>
<td>DEC PDP11</td>
</tr>
<tr>
<td>IN, AL, KBD</td>
<td>Load a byte from in port KBD to accumulator</td>
<td>Intel Pentium</td>
</tr>
<tr>
<td>LEA.L (A0), A2</td>
<td>Load the address pointed to by A0 into A2</td>
<td>M6800</td>
</tr>
</tbody>
</table>

- Lots of variation, even with one instruction type
## Tbl 2.2 Examples of ALU Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULF A, B, C</td>
<td>multiply the 32-bit floating point values at mem loc’ns. A and B, store at C</td>
<td>VAX11</td>
</tr>
<tr>
<td>nabs r3, r1</td>
<td>Store abs value of r1 in r3</td>
<td>PPC601</td>
</tr>
<tr>
<td>ori $2, $1, 255</td>
<td>Store logical OR of reg $1 with 255 into reg $2</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>DEC R2</td>
<td>Decrement the 16-bit value stored in reg R2</td>
<td>DEC PDP11</td>
</tr>
<tr>
<td>SHL AX, 4</td>
<td>Shift the 16-bit value in reg AX left by 4 bit pos’ns.</td>
<td>Intel 8086</td>
</tr>
</tbody>
</table>

- Notice again the complete dissimilarity of both syntax and semantics.
# Tbl 2.3 Examples of Branch Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Meaning</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLSS A, Tgt</td>
<td>Branch to address Tgt if the least significant bit of mem loc’n. A is set (i.e. = 1)</td>
<td>VAX11</td>
</tr>
<tr>
<td>bun r2</td>
<td>Branch to location in R2 if result of previous floating point computation was Not a Number (NAN)</td>
<td>PPC601</td>
</tr>
<tr>
<td>beq $2, $1, 32</td>
<td>Branch to location (PC + 4 + 32) if contents of $1 and $2 are equal</td>
<td>MIPS R3000</td>
</tr>
<tr>
<td>SOB R4, Loop</td>
<td>Decrement R4 and branch to Loop if R4 ≠ 0</td>
<td>DEC PDP11</td>
</tr>
<tr>
<td>JCXZ Addr</td>
<td>Jump to Addr if contents of register CX ≠ 0</td>
<td>Intel 8086</td>
</tr>
</tbody>
</table>
CPU Registers Associated with Flow of Control—Branch Instructions

- Program counter usually locates next instruction
- Condition codes may control branch
- Branch targets may be separate registers

Processor State

Program Counter

Condition Codes

Branch Targets
HLL Conditionals Implemented by Control Flow Change

- Conditions are computed by arithmetic instructions
- Program counter is changed to execute only instructions associated with true conditions

```c
if NUM==5 then SET=7
```

```assembly
CMP.W #5, NUM ;the comparison
BNE L1 ;conditional branch
MOV.W #7, SET ;action if true
L1 ... ;action if false
```
3-, 2-, 1-, & 0-Address ISAs

- The classification is based on arithmetic instructions that have two operands and one result.
- The key issue is “how many of these are specified by memory addresses, as opposed to being specified implicitly.”
- A 3-address instruction specifies memory addresses for both operands and the result \( R \leftarrow \text{Op1 op Op2} \).
- A 2-address instruction overwrites one operand in memory with the result \( \text{Op2} \leftarrow \text{Op1 op Op2} \).
- A 1-address instruction has a processor, called the accumulator register, to hold one operand & the result (no addr. needed) \( \text{Acc} \leftarrow \text{Acc op Op1} \).
- A 0-address + uses a CPU register stack to hold both operands and the result \( \text{TOS} \leftarrow \text{TOS op SOS} \) (where TOS is Top Of Stack, SOS is Second On Stack).
- The 4-address instruction, hardly ever seen, also allows the address of the next instruction to specified explicitly.
Fig 2.2 The 4-Address Machine and Instruction Format

- Explicit addresses for operands, result, & next instruction
- Example assumes 24-bit addresses
  - Discuss: size of instruction in bytes

Instruction format

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>24</th>
<th>24</th>
<th>24</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Res Addr</td>
<td>Op1 Addr</td>
<td>Op2 Addr</td>
<td>Nexti Addr</td>
<td></td>
</tr>
</tbody>
</table>

Which operation
Where to put result
Where to find operands
Where to find next instruction

Fig 2.3 The 3-Address Machine and Instruction Format

- Address of next instruction kept in processor state register—the PC (except for explicit branches/jumps)
- Rest of addresses in instruction
  - Discuss: savings in instruction word size
Fig 2.4 The 2-Address Machine and Instruction Format

- Result overwrites Operand 2
- Needs only 2 addresses in instruction but less choice in placing data
Fig 2.5 1-Address Machine and Instruction Format

Need instructions to load and store operands:
LDA OpAddr
STA OpAddr

- Special CPU register, the accumulator, supplies 1 operand and stores result
- One memory address used for other operand

Instruction format

<table>
<thead>
<tr>
<th>Bits</th>
<th>8</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Op1Addr</td>
<td></td>
</tr>
</tbody>
</table>

Where to find operand2, and where to put result

add Op1 (Acc ← Acc + Op1)
Fig 2.6 The 0-Address, or Stack, Machine and Instruction Format

- Uses a push-down stack in CPU
- Arithmetic uses stack for both operands and the result
- Computer must have a 1-address instruction to push and pop operands to and from the stack
Example 2.1 Expression Evaluation for 3-, 2-, 1-, and 0-Address Machines

Evaluate: \[ a = (b+c)*d - e \]

<table>
<thead>
<tr>
<th>3-address</th>
<th>2-address</th>
<th>1-address</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>add a, b, c</td>
<td>load a, b</td>
<td>load b</td>
<td>push b</td>
</tr>
<tr>
<td>mpy a, a, d</td>
<td>add a, c</td>
<td>add c</td>
<td>push c</td>
</tr>
<tr>
<td>sub a, a, e</td>
<td>mpy a, d</td>
<td>mpy d</td>
<td>add</td>
</tr>
<tr>
<td>sub a, e</td>
<td>sub e</td>
<td>push d</td>
<td></td>
</tr>
<tr>
<td></td>
<td>store a</td>
<td>mpy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>push e</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sub</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pop a</td>
<td></td>
</tr>
</tbody>
</table>

- Number of instructions & number of addresses both vary
- Discuss as examples: size of code in each case
Fig 2.7 General Register Machine and Instruction Formats

- It is the most common choice in today’s general-purpose computers
- *Which* register is specified by small “address” (3 to 6 bits for 8 to 64 registers)
- Load and store have one long & one short address: 1-1/2 addresses
- Arithmetic instruction has 3 “half” addresses