Chapter 11
Instruction Sets:
Addressing Modes and Formats
Contents

• Addressing
• Pentium and PowerPC addressing modes
• Instruction formats
• Pentium and PowerPC instruction formats
11.1 Addressing

- How to specify the locations of operands?
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Stack
Immediate Addressing

Instruction

| Operand |

- Operand is part of instruction
  - ADD 5
    - Add 5 to contents of accumulator
    - 5 is operand
- No memory reference to fetch data
- Size of the number is restricted to the size of the address field
Direct Addressing

- Address field contains the address of operand
  - $EA = A$
    - Effective address is equal to the address field

- ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand

- Single memory reference to access data

- No additional calculations to get effective address

- Address space is limited to the size of the address field
Direct Addressing
Indirect Addressing

- Memory cell pointed to by address field contains the address of the operand
- \( EA = (A) \)
  - Look in A, find address \((A)\) and look there for operand
- ADD \((A)\)
  - Add contents of cell pointed to by contents of A to accumulator
- Larger address space is possible
  - \(2^n\) where \(n = \) word length
- May be nested, multilevel, cascaded
  - \(EA = (((A)))\)
- Multiple memory accesses to find operand
Indirect Addressing
Register Addressing

- Operand is held in register
- EA = R

- Advantages
  - Only a small address field is needed
    - Shorter instructions
  - No memory access
    - Fast execution is possible

- Very limited address space
Register Addressing

Instruction

R

Operand

Registers
Register Indirect Addressing

- Operand is in memory cell pointed to by contents of register R
- EA = (R)
- Large address space: $2^n$
- One fewer memory access than indirect addressing
Register Indirect Addressing
Displacement Addressing

• Combines the capabilities of direct addressing and register indirect addressing

• EA = A + (R)

• Instruction has two address fields
  — A = base value
  — R = register that holds displacement
  — or vice versa

• Common uses of displacement addressing
  — Relative addressing
  — Base-register addressing
  — Indexing
Displacement Addressing
Relative Addressing

- \( R = PC \) (Program Counter)
- \( EA = A + (PC) \)

Current instruction address is added to the address field to produce the EA
- Address field is treated as a twos complement number for this operation
- Effective address is a displacement relative to the address of the instruction

- LOOP: ADD BX, AX
  
  ..... 
  
  JNZ LOOP
Base-Register Addressing

- \( EA = A + (R) \)
- \( A \) holds displacement
- \( R \) holds pointer to base address
  - \( R \) may be explicit or implicit
- Segment registers in 80x86
  - Offset address is added to the contents of segment register
Indexing

- $EA = A + (R)$
- $A = \text{base}$
- $R = \text{positive displacement}$
- Opposite of base-register addressing
- Good for accessing arrays
  - $EA = A + (R)$
  - $(R) \leftarrow (R) + 1$
Stack Addressing

- Operand is (implicitly) on top of stack
- ADD
  - Pop top two items from stack, add, push the result onto stack
Stack Addressing

Instruction

Implicit

Top of Stack Register
# Summary of Addressing Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
<th>Principal Advantage</th>
<th>Principal Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Operand = A</td>
<td>No memory reference</td>
<td>Limited operand magnitude</td>
</tr>
<tr>
<td>Direct</td>
<td>EA = A</td>
<td>Simple</td>
<td>Limited address space</td>
</tr>
<tr>
<td>Indirect</td>
<td>EA = (A)</td>
<td>Large address space</td>
<td>Multiple memory references</td>
</tr>
<tr>
<td>Register</td>
<td>EA = R</td>
<td>No memory reference</td>
<td>Limited address space</td>
</tr>
<tr>
<td>Register indirect</td>
<td>EA = (R)</td>
<td>Large address space</td>
<td>Extra memory reference</td>
</tr>
<tr>
<td>Displacement</td>
<td>EA = A + (R)</td>
<td>Flexibility</td>
<td>Complexity</td>
</tr>
<tr>
<td>Stack</td>
<td>EA = top of stack</td>
<td>No memory reference</td>
<td>Limited applicability</td>
</tr>
</tbody>
</table>
11.2 Pentium Addressing Modes

- Virtual or effective address is offset into segment
  - Starting address plus offset gives linear address
  - This goes through page translation if paging is enabled
- 9 addressing modes available
  - Immediate
  - Register operand
  - Displacement: offset in a segment
  - Base: same as register indirect addressing
  - Base with displacement
  - Scaled index with displacement: scale factor is used
    - scale factor of 2 can be used to index an array of 16-bit integers
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative
    - used in transfer-of-control instructions
Figure 11.2  Pentium Addressing Mode Calculation
<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>Operand = A</td>
</tr>
<tr>
<td>Register Operand</td>
<td>LA = R</td>
</tr>
<tr>
<td>Displacement</td>
<td>LA = (SR) + A</td>
</tr>
<tr>
<td>Base</td>
<td>LA = (SR) + (B)</td>
</tr>
<tr>
<td>Base with Displacement</td>
<td>LA = (SR) + (B) + A</td>
</tr>
<tr>
<td>Scaled Index with Displacement</td>
<td>LA = (SR) + (I) \times S + A</td>
</tr>
<tr>
<td>Base with Index and Displacement</td>
<td>LA = (SR) + (B) + (I) + A</td>
</tr>
<tr>
<td>Base with Scaled Index and Displacement</td>
<td>LA = (SR) + (I) \times S + (B) + A</td>
</tr>
<tr>
<td>Relative</td>
<td>LA = (PC) + A</td>
</tr>
</tbody>
</table>

**Pentium II Addressing Modes**

- **LA** = linear address
- **(X)** = contents of X
- **SR** = segment register
- **PC** = program counter
- **A** = contents of an address field in the instruction
- **R** = register
- **B** = base register
- **I** = index register
- **S** = scaling factor
PowerPC Addressing Modes

- Uses a simple set of addressing modes
  - Load/store addressing
    - Indirect: $EA = (BR) + D$
      + Instruction includes 16 bit displacement to be added to base register
      + Can replace base register content with new address
    - Indirect indexed: $EA = (BR) + (IR)$
      + Instruction references base register and index register
      + $EA$ is sum of contents of both registers
  - Branch addressing
    - Absolute: $EA = I$
    - Relative: $EA = (PC) + I$
    - Indirect: $EA = (L/CR)$
  - Arithmetic instructions
    - Operands in registers or part of instruction
    - Register addressing only for floating point computation
## PowerPC Addressing Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load/Store Addressing</strong></td>
<td></td>
</tr>
<tr>
<td>Indirect</td>
<td>$EA = (BR) + D$</td>
</tr>
<tr>
<td>Indirect Indexed</td>
<td>$EA = (BR) + (IR)$</td>
</tr>
<tr>
<td><strong>Branch Addressing</strong></td>
<td></td>
</tr>
<tr>
<td>Absolute</td>
<td>$EA = I$</td>
</tr>
<tr>
<td>Relative</td>
<td>$EA = (PC) + I$</td>
</tr>
<tr>
<td>Indirect</td>
<td>$EA = (L/CR)$</td>
</tr>
<tr>
<td><strong>Fixed-Point Computation</strong></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>$EA = GPR$</td>
</tr>
<tr>
<td>Immediate</td>
<td>Operand = I</td>
</tr>
<tr>
<td><strong>Floating-Point Computation</strong></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>$EA = FPR$</td>
</tr>
</tbody>
</table>

$EA$ = effective address  
$(X)$ = contents of $X$  
$BR$ = base register  
$IR$ = index register  
$L/CR$ = link or count register  
$GPR$ = general-purpose register  
$FPR$ = floating-point register  
$D$ = displacement  
$I$ = immediate value  
$PC$ = program counter
PowerPC Memory Operand Addressing Modes

(a) Indirect Addressing

(b) Indirect Indexed Addressing
11.3 Instruction Formats

- Layout of bits in an instruction
  - Includes opcode
  - Includes (implicit or explicit) operand(s)

- Usually more than one instruction format in an instruction set

- Design issues
  - Instruction length
  - Allocation of bits
  - Fixed/Variable-length instructions
Instruction Length

- Affects and affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed

- Trade off between powerful instruction repertoire and saving space
  - More opcodes and operands make shorter program possible
  - More addressing modes and address fields make larger address space possible
  - But longer instruction length may be wasteful
Allocation of Bits

- Design factors for allocation of bits
  - Number of addressing modes
    - If we need to specify it explicitly, some bits are needed
  - Number of operands
  - Register versus memory
    - Only a few bits are needed to specify the register
    - Most machines today have at least 32 general purpose registers
  - Number of register sets
    - General-purpose set + specialized set
  - Address range
    - Indirect addressing is favored to address larger space
  - Address granularity
    - Byte or word addressing
    - Byte addressing is convenient for character manipulation but requires more address bits
PDP-8

- One of the simplest instruction designs
  - 12-bit instructions on 12-bit words
    - 3-bit opcode and three type of instructions
    - Memory is divided into pages of $2^7$ words each
  - Single GPR, accumulator

- Instruction format
  - Memory reference instructions: opcodes 0 through 5
    - Includes a page bit and indirect bit
  - I/O instructions: opcode 6
    - 6 bits used to select one of 64 devices
    - 3 bits to specify a particular command
  - Register reference instructions: opcode 7
    - Remaining bits are used to encode additional operations
## PDP-8 Instruction Format

### Memory Reference Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>D/I</th>
<th>Z/C</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

### Input/Output Instructions

<table>
<thead>
<tr>
<th>Device</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Register Reference Instructions

#### Group 1 Microinstructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CLA</th>
<th>CLL</th>
<th>CMA</th>
<th>CML</th>
<th>RAR</th>
<th>RAL</th>
<th>BSW</th>
<th>IAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

#### Group 2 Microinstructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CLA</th>
<th>SMA</th>
<th>SZA</th>
<th>SNL</th>
<th>RSS</th>
<th>OSR</th>
<th>HLT</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

#### Group 3 Microinstructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>CLA</th>
<th>MQA</th>
<th>MQL</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

- **D/I** = Direct/Indirect address
- **Z/C** = Page 0 or Current page
- **CLA** = Clear Accumulator
- **CLL** = Clear Link
- **CMA** = Complement Accumulator
- **CML** = Complement Link
- **RAR** = Rotate Accumulator Right
- **RAL** = Rotate Accumulator Left
- **BSW** = Byte Swap
- **IAC** = Increment ACCumulator
- **SMA** = Skip on Minus Accumulator
- **SZA** = Skip on Zero Accumulator
- **SNL** = Skip on Nonzero Link
- **RSS** = Reverse Skip Sense
- **OSR** = Or with Switch Register
- **HLT** = HA LT
- **MQA** = Multiplier Quotient into Accumulator
- **MQL** = Multiplier Quotient Load
PDP-10

• Designed to be a large-scale time-shared system

• Design principles
  — Orthogonality
    – Other elements of an instruction are independent of the opcode
    + an address is always computed in the same way, independent of the opcode
  — Completeness
    – Each arithmetic data type should have a complete and identical set of operations
  — Direct addressing
PDP-10

- Instruction format
  - 36-bit word length and 36-bit instruction length

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Register</th>
<th>I</th>
<th>Index Register</th>
<th>Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>9</td>
<td>12</td>
<td>14 17 18 35</td>
</tr>
</tbody>
</table>

I = indirect bit

- 9 bits for opcodes
  - 365 instructions are defined

- Most instructions have two addresses
  - One is one of 16 GPRs
  - The other is 18-bit memory address
PDP-11

- 16-bit minicomputer
  - Has variable-length instructions
  - Has 16 GPRs, including PC and SP

- Instruction format
  - 13 different formats are used
    - Zero-, one-, and two-address instructions
  - Opcode lengths vary from 4 to 16 bits
  - 6 bits for register reference
    - 3 to identify the register and 3 to identify modes
  - Instructions are usually 16-bits long
    - For some instructions, one or two memory addresses are appended
# PDP-11 Instruction Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>4</td>
</tr>
<tr>
<td>Source</td>
<td>6</td>
</tr>
<tr>
<td>Destination</td>
<td>6</td>
</tr>
<tr>
<td>Opcode</td>
<td>7</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>Source</td>
<td>6</td>
</tr>
<tr>
<td>Opcode</td>
<td>8</td>
</tr>
<tr>
<td>Offset</td>
<td>8</td>
</tr>
<tr>
<td>Opcode</td>
<td>8</td>
</tr>
<tr>
<td>FP</td>
<td>2</td>
</tr>
<tr>
<td>Destination</td>
<td>6</td>
</tr>
<tr>
<td>Opcode</td>
<td>10</td>
</tr>
<tr>
<td>Destination</td>
<td>6</td>
</tr>
<tr>
<td>Opcode</td>
<td>12</td>
</tr>
<tr>
<td>CC</td>
<td>4</td>
</tr>
<tr>
<td>Opcode</td>
<td>13</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>Memory Address</td>
<td>16</td>
</tr>
<tr>
<td>Opcode</td>
<td>14</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>Source</td>
<td>6</td>
</tr>
<tr>
<td>Memory Address</td>
<td>16</td>
</tr>
<tr>
<td>Opcode</td>
<td>15</td>
</tr>
<tr>
<td>FP</td>
<td>2</td>
</tr>
<tr>
<td>Source</td>
<td>6</td>
</tr>
<tr>
<td>Memory Address</td>
<td>16</td>
</tr>
<tr>
<td>Opcode</td>
<td>16</td>
</tr>
<tr>
<td>Destination</td>
<td>6</td>
</tr>
<tr>
<td>Memory Address</td>
<td>16</td>
</tr>
<tr>
<td>Opcode</td>
<td>17</td>
</tr>
<tr>
<td>Source</td>
<td>6</td>
</tr>
<tr>
<td>Destination</td>
<td>6</td>
</tr>
<tr>
<td>Memory Address 1</td>
<td>16</td>
</tr>
<tr>
<td>Memory Address 2</td>
<td>16</td>
</tr>
</tbody>
</table>

Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field
VAX

- Design principles
  - All instructions should have the natural number of operands
  - All operands should have the same generality in specification
- Result is a highly variable instruction format
  - Instruction consists of a 1- or 2-byte opcode followed by from zero to six operand specifiers
    - Instructions from 1 to 37 bytes long
  - Instruction begins with a 1-byte opcode
    - FD and FF indicate an extended opcode
    - Second byte specify opcode
VAX

- Instruction with 6 operands
  - ADDP6 OP1, OP2, OP3, OP4, OP5, OP6
    - Instruction for adding two packed decimal numbers
    - OP1 and OP2 specify the length and starting address of one decimal string
    - OP3 and OP4 specify second string
    - Result is stored in a location specified by OP5 and OP6
11.4 Pentium Instruction Formats

- Instruction consists of
  - 0-4 optional prefixes
  - 1-2 byte opcode
  - Optional address specifier
    - Consists of ModR/m byte and Scale Index byte
  - Optional displacement
  - Optional immediate field
Pentium Instruction Formats

• Prefix bytes
  — Instruction prefixes
    – LOCK or one of repeat prefixes
    – LOCK is used to ensure exclusive use of shared memory in multiprocessor environments
    – REP, REPE, REPZ, REPNE, and REPNZ
      + Specify repeated operation on strings
      + Repeat until counter in CX goes zero or until the condition is met
  — Segment override
  — Address size
    – Switches between 32-bit and 16-bit address generation
  — Operand size
    – Switches between 32-bit and 16-bit operands
Pentium Instruction Formats

- Instruction
  - Opcode
  - ModR/m
    - Specify whether an operand is in a register or in memory
  - SIB
    - Specify fully the addressing mode
  - Displacement
    - When used, 8-, 16-, or 32-bit displacement field is added
  - Immediate
    - When used, 8-, 16-, or 32-bit operand is provided
Pentium Instruction Formats

<table>
<thead>
<tr>
<th>Instruction prefix</th>
<th>Segment override</th>
<th>Operand size override</th>
<th>Address size override</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>0 or 1</td>
<td>0 or 1</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

- Instruction prefixes: 0, 1, 2, 3, or 4 bytes
- Opcode: 1 or 2
- ModR/M: 0 or 1
- SIB: 0 or 1
- Displacement: 0, 1, 2, or 4
- Immediate: 0, 1, 2, or 4

<table>
<thead>
<tr>
<th>Mod</th>
<th>Reg/Opcode</th>
<th>R/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scale</th>
<th>Index</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
PowerPC Instruction Formats

- Instruction formats
  - All instructions are 32 bits long and follow a regular format
  - First 6 bits specify the operation
    - May be extended elsewhere in the instruction for subcases
  - 5 bits for register references
# PowerPC Instruction Formats

<table>
<thead>
<tr>
<th>Branch Condition</th>
<th>Long Immediate</th>
<th>A</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Br Conditional</td>
<td>Options</td>
<td>CR Bit</td>
<td>Branch Displacement</td>
</tr>
<tr>
<td>Br Conditional</td>
<td>Options</td>
<td>CR Bit</td>
<td>Indirect through Link or Count Register</td>
</tr>
</tbody>
</table>

(a) Branch instructions

<table>
<thead>
<tr>
<th>CR</th>
<th>Dest Bit</th>
<th>Source Bit</th>
<th>Source Bit</th>
<th>Add, OR, XOR, etc.</th>
<th>/</th>
</tr>
</thead>
</table>

(b) Condition register logical instructions

<table>
<thead>
<tr>
<th>Ld/St Indirect</th>
<th>Dest Register</th>
<th>Base Register</th>
<th>Index Register</th>
<th>Displacement</th>
<th>Size, Sign, Update</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ld/St Indirect</td>
<td>Dest Register</td>
<td>Base Register</td>
<td>Index Register</td>
<td>Displacement</td>
<td>Size, Sign, Update</td>
<td>/</td>
</tr>
<tr>
<td>Ld/St Indirect</td>
<td>Dest Register</td>
<td>Base Register</td>
<td></td>
<td>Displacement</td>
<td>XO *</td>
<td></td>
</tr>
</tbody>
</table>

(c) Load/store instructions
# PowerPC Instruction Formats

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Dest Register</th>
<th>Src Register</th>
<th>Src Register</th>
<th>O</th>
<th>Add, Sub, etc.</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, Sub, etc.</td>
<td>Dest Register</td>
<td>Src Register</td>
<td>Signed Immediate Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Src Register</td>
<td>ADD, OR, XOR, etc.</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>AND, OR, etc.</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Unsigned Immediate Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rotate</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Shift Amt</td>
<td>Mask Begin</td>
<td>Mask End</td>
<td>R</td>
</tr>
<tr>
<td>Rotate or Shift</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Src Register</td>
<td>Shift Type or Mask</td>
<td></td>
<td>R</td>
</tr>
<tr>
<td>Rotate</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Shift Amt</td>
<td>Mask</td>
<td>XO</td>
<td>S R</td>
</tr>
<tr>
<td>Rotate</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Src Register</td>
<td>Mask</td>
<td>XO</td>
<td>R</td>
</tr>
<tr>
<td>Shift</td>
<td>Src Register</td>
<td>Dest Register</td>
<td>Shift Type or Mask</td>
<td></td>
<td>S R</td>
<td></td>
</tr>
</tbody>
</table>

(d) Integer arithmetic, logical, and shift/rotate instructions

| Flt sgl/dbl | Dest Register | Src Register | Src Register | Src Register | Fadd, etc. | R |
|------------|---------------|--------------|--------------|--------------|------------|

(e) Floating-point arithmetic instructions

- A = Absolute or PC relative
- L = Link or subroutine
- O = Record overflow in XER
- R = Record condition in CR1
- XO = Opcode extension
- S = Part of shift amount field
- * = 64-bit implementation only