COMPUTER HARDWARE

Instruction Set Architecture
Overview

- Computer architecture
- Operand addressing
  - Addressing architecture
  - Addressing modes
- Elementary instructions
  - Data transfer instructions
  - Data manipulation instructions
    - Floating point computations
  - Program control instructions
    - Program interrupt and exceptions
Overview

**Computer Architecture**
- Instruction set architecture
  - A set of hardware-implemented instructions, the symbolic name and the binary code format of each instruction

**Organization**
- Structures such as datapath, control units, memories, and the busses that interconnect them

**Hardware**
- The logic, the electronic technology employed, the various physical design aspects of the computer

**Example ISAs (Instruction Set Architectures)**
- RISC (Reduced Instruction Set Computer)
  - Digital Alpha
  - Sun Sparc
  - MIPS RX000
  - IBM PowerPC
  - HP PA/RISC
- CISC (Complex Instruction Set Computer)
  - Intel x86
  - Motorola 68000
  - DEC VAX
- VLIW (Very Large Instruction Word)
  - Intel Itanium

**Instruction Set Architecture**
- A processor is specified completely by its instruction set architecture (ISA)
- Each ISA will have a variety of instructions and instruction formats, which will be interpreted by the computer
- An instruction represents the smallest indivisible unit of computation. It is a string of bits grouped into different numbers and size of substrings (fields)
  - Operation code (opcode): the operation to be performed
  - Address field: where we can find the operands needed for that operation
  - Mode field: the information given in the address field
  - Other fields: constant immediate operand or shift

```plaintext
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw$15, 0($2)
lw$16, 4($2)
sw $16, 0($2)
sw $15, 4($2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

**Machine Interpretation**
- ALUOP[0:3] <= InstReg[9:11] & MASK
Computer Architecture

- **Instruction set architecture**
  - A set of hardware-implemented instructions, the symbolic name and the binary code format of each instruction

- **Organization**
  - Structures such as datapath, control units, memories, and the busses that interconnect them

- **Hardware**
  - The logic, the electronic technology employed, the various physical design aspects of the computer
Example ISAs (Instruction Set Architectures)

- **RISC (Reduced Instruction Set Computer)**
  - Digital Alpha
  - Sun Sparc
  - MIPS RX000
  - IBM PowerPC
  - HP PA/RISC

- **CISC (Complex Instruction Set Computer)**
  - Intel x86
  - Motorola 68000
  - DEC VAX

- **VLIW (Very Large Instruction Word)**
  - Intel Itanium
A processor is specified completely by its *instruction set architecture (ISA)*

Each ISA will have a variety of instructions and instruction formats, which will be interpreted by the processor’s control unit and executed in the processor’s datapath.

An instruction represents the smallest indivisible unit of computation. It is a string of bits grouped into different numbers and size of *substrings (fields)*

- **Operation code (opcode):** the operation to be performed
- **Address field:** where we can find the operands needed for that operation
- **Mode field:** how to derive the data’s effective address from the information given in the address field
- **Other fields:** constant immediate operand or shift
Computer Operation Cycle

- **Instruction Fetch**: Obtain instruction from program storage
- **Instruction Decode**: Determine required actions and instruction size
- **Operand Fetch**: Locate and obtain operand data
- **Execute**: Compute result value or status
- **Result Store**: Deposit results in storage for later use
- **Next Instruction**: Determine successor instruction

**Register Set**
- Programmer accessible registers (R₀ to R₇ in previous multi-cycle computer)
- Other registers
  - Registers in the register file accessible only to microprograms (R₈ to R₁₅)
  - Instruction registers (IR)
  - Program counter (PC)
  - Pipeline registers
  - Processor status register (PSR: CVNZ state)
  - Stack pointer (SP)

**Operand Addressing**
- **Operand**: register value, memory content, or immediate
- **Explicit address**: address field in the instruction
- **Implied address**: the location of operand is specified by the opcode or other operand address

**Three Address Instructions**
- Example: X = (A + B)(C + D)
- Operands are in memory address symbolized by the letters A, B, C, D, result stored memory address of X

**ADD T1, A, B**

**ADD T2, C, D**

**MUX X, T1, T2**
M[X] ← M[T1] X M[T2]

**OR**

**ADD R1, A, B**
R1 ← M[A] + M[B]

**ADD R2, C, D**
R2 ← M[C] + M[D]

**MUX X, R1, R2**
M[X] ← R1 X R2

- **+: Short program, 3 instructions
- **-**: Binary coded instruction require more bits to specify three addresses
Register Set

- Programmer accessible registers (R0 to R7 in previous multi-cycle computer)

- Other registers
  - Registers in the register file accessible only to microprograms (R8 to R15)
  - Instruction registers (IR)
  - Program counter (PC)
  - Pipeline registers
  - Processor status register (PSR: CVNZ state)
  - Stack pointer (SP)
Operand Addressing

- **Operand**: register value, memory content, or immediate
- **Explicit address**: address field in the instruction
- **Implied address**: the location of operand is specified by the opcode or other operand address
Three Address Instructions

- Example: \( X = (A+B)(C+D) \)
- Operands are in memory address symbolized by the letters A, B, C, D, result stored memory address of X

\[
\begin{align*}
\text{ADD T1, A, B} & \quad M[T1] \leftarrow M[A] + M[B] \\
\text{ADD T2, C, D} & \quad M[T2] \leftarrow M[C] + M[D] \\
\text{MUX X, T1, T2} & \quad M[X] \leftarrow M[T1] \times M[T2] \\
\end{align*}
\]

**OR**

\[
\begin{align*}
\text{ADD R1, A, B} & \quad R1 \leftarrow M[A] + M[B] \\
\text{ADD R2, C, D} & \quad R2 \leftarrow M[C] + M[D] \\
\text{MUX X, R1, R2} & \quad M[X] \leftarrow R1 \times R2 \\
\end{align*}
\]

- +: Short program, 3 instructions
- -: Binary coded instruction require more bits to specify three addresses
Two Address Instructions

- The first operand address also serves as the implied address for the result.

### Instructions

- **MOVE T1, A**
  
  \[ M[T1] \leftarrow M[A] \]

- **ADD T1, B**
  
  \[ M[T1] \leftarrow M[T1] + M[B] \]

- **MOVE X, C**
  
  \[ M[X] \leftarrow M[C] \]

- **ADD X, D**
  
  \[ M[X] \leftarrow M[X] + M[D] \]

- **MUX X, T1**
  
  \[ M[X] \leftarrow M[X] \times M[T1] \]

### 5 instructions
### One Address Instructions

- **Implied address:** a register called *an accumulator ACC* for one operand and the result, *single-accumulator architecture*

  - **LD** A \( \rightarrow \text{ACC} \leftarrow \text{M}[A] \)
  - **ADD** B \( \rightarrow \text{ACC} \leftarrow \text{ACC} + \text{M}[B] \)
  - **ST** X \( \rightarrow \text{M}[X] \leftarrow \text{ACC} \)
  - **LD** C \( \rightarrow \text{ACC} \leftarrow \text{M}[C] \)
  - **ADD** D \( \rightarrow \text{ACC} \leftarrow \text{ACC} + \text{M}[D] \)
  - **MUX** X \( \rightarrow \text{ACC} \leftarrow \text{ACC} \times \text{M}[X] \)
  - **ST** X \( \rightarrow \text{M}[X] \leftarrow \text{ACC} \)

- All operations are between the ACC register and a memory operand

### Zero Address Instructions

- Use stack (FILO):
  - **ADD** TOS \( \rightarrow TOS + TOS - 1 \)
  - **PUSH** X \( \rightarrow \text{M}[X] \)
  - **POP** X \( \rightarrow \text{M}[X] \leftarrow \text{TOS} \)
  - **PUSH** A \( \rightarrow \text{M}[A] \)
  - **PUSH** B \( \rightarrow \text{M}[B] \)
  - **ADD** TOS \( \rightarrow TOS + TOS - 1 \)
  - **PUSH** C \( \rightarrow \text{M}[C] \)
  - **PUSH** D \( \rightarrow \text{M}[D] \)
  - **ADD** TOS \( \rightarrow TOS + TOS - 1 \)
  - **MUX** TOS \( \rightarrow TOS \times TOS - 1 \)
  - **POP** X \( \rightarrow \text{M}[X] \leftarrow \text{TOS} \)

### Stack Instructions

- **Push:**
  - \( \text{SP} = 101 \)
  - \( \text{SP} \rightarrow \text{SP} - 1; \text{TOS} \rightarrow \text{R1} \)

- **Pop:**
  - \( \text{R1} \rightarrow \text{TOS}; \text{SP} \rightarrow \text{SP} + 1 \)

### Addressing Architecture

- Defines:
  - Restriction on the number of memory addresses in instructions
  - Number of operands

- Two kinds of addressing architecture:
  - Memory-to-memory architecture
    - Only one register - PC
    - All operands from memory, and results to memory
    - Many memory accesses
  - Register-to-register (load/store) architecture
    - Restrict only one memory address to load/store types, all other operations are between registers

- **LD** R1, A \( \rightarrow \text{R1} \leftarrow \text{M}[A] \)
- **LD** R2, B \( \rightarrow \text{R2} \leftarrow \text{M}[B] \)
- **ADD** R3, R1, R2 \( \rightarrow \text{R3} \leftarrow \text{R1} + \text{R2} \)
- **LD** R1, C \( \rightarrow \text{R1} \leftarrow \text{M}[C] \)
- **LD** R2, D \( \rightarrow \text{R2} \leftarrow \text{M}[D] \)
- **ADD** R1, R1, R2 \( \rightarrow \text{R1} \leftarrow \text{R1} + \text{R2} \)
- **MUL** R1, R1, R3 \( \rightarrow \text{R1} \leftarrow \text{R1} \times \text{R3} \)
- **ST** X, R1 \( \rightarrow \text{M}[X] \leftarrow \text{R1} \)

### Addressing Modes

- **Address field:** contains the information needed to determine the location of the operands and the result of an operation
- **Addressing mode:** specifies how to interpret the information within this address field, how to compute the actual or effective address of the data needed.
- Availability of a variety of addressing modes lets programmers write more efficient code
Zero Address Instructions

- Use stack (FILO):
  - ADD             TOS ← TOS + TOS_{-1}
  - PUSH X          TOS ← M[X]
  - POP X           M[X] ← TOS

  PUSH A            TOS ← M[A]
  PUSH B            TOS ← M[B]
  ADD               TOS ← TOS + TOS_{-1}
  PUSH C            TOS ← M[C]
  PUSH D            TOS ← M[D]
  ADD               TOS ← TOS + TOS_{-1}
  MUX               TOS ← TOS × TOS_{-1}
  POP X             M[X] ← TOS

- Data manipulation operations: between the stack elements
- Transfer operations: between the stack and the memory
Stack Instructions

- Push: \( \text{SP} \leftarrow \text{SP}-1; \ TOS \leftarrow \text{R1} \)
- Pop: \( \text{R1} \leftarrow \text{TOS}; \ SP \leftarrow \text{SP} +1 \)
Stack Architecture

- The infix architecture
  \[(A + B) \times C + (D \times E)\]
- Reverse Polish Notation (RPN)
  \[A \ B \ + \ C \times \ D \ E \ \times \ +\]
Addressing Architecture

- Defines:
  - Restriction on the number of memory addresses in instructions
  - Number of operands

- Two kinds of addressing architecture:
  - Memory-to-memory architecture
    - Only one register - PC
    - All operands from memory, and results to memory
    - Many memory accesses
  - Register-to-register (load/store) architecture
    - Restrict only one memory address to load/store types, all other operations are between registers

```
LD  R1, A       R1 ← M[A]
LD  R2, B       R2 ← M[B]
ADD R3, R1, R2  R3 ← R1 + R2
LD  R1, C       R1 ← M[C]
LD  R2, D       R2 ← M[D]
ADD R1, R1, R2  R1 ← R1 + R2
MUL R1, R1, R3  R1 ← R1 × R3
ST  X, R1       M[X] ← R1
```

- Addressing Modes
  - Implied mode
  - Immediate mode (operand)
  - Register mode
  - Register-indirect mode
  - Direct addressing mode
  - Indirect Addressing mode
  - PC-relative addressing
Addressing Modes

- **Address field**: contains the information needed to determine the location of the operands and the result of an operation.

- **Addressing mode**: specifies how to interpret the information within this address field, how to compute the **actual** or **effective** address of the data needed.

- Availability of a variety of addressing modes lets programmers write more efficient code.
Addressing Modes

- **Implied** mode - implied in the opcode, such as stack, accumulator
- **Immediate** mode (operand) - \( a = 0x0801234 \)
- **Register** mode - \( a = R[b] \)
- **Register-indirect** mode – \( a = M[R[b]] \)
- **Direct** addressing mode - \( a = M[0x0013df8] \)
- **Indirect** Addressing mode - \( a = M[M[0x0013df8]] \)
- **PC-relative** addressing – branch etc. (offset + PC)
- **Indexed** addressing - \( a = b[1] \)
Demonstrating Direct Addressing

Memory

<table>
<thead>
<tr>
<th>250</th>
<th>Opcode</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>ADRS</td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>Next instruction</td>
<td></td>
</tr>
</tbody>
</table>

Program

Data

PC = 250

ACC

Opcode: Load ACC
Mode: Direct address
ADRS: 500
Operation: ACC ← 800
Example

Opcode: Load to ACC, ADRS or NBR=500

<table>
<thead>
<tr>
<th>Memory</th>
<th>Opcode</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>251</td>
<td></td>
<td>ADRS or NBR = 500</td>
</tr>
<tr>
<td>252</td>
<td></td>
<td>Next instruction</td>
</tr>
<tr>
<td>400</td>
<td>400</td>
<td>700</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>800</td>
</tr>
<tr>
<td>752</td>
<td>752</td>
<td>600</td>
</tr>
<tr>
<td>800</td>
<td>800</td>
<td>300</td>
</tr>
<tr>
<td>900</td>
<td>900</td>
<td>200</td>
</tr>
</tbody>
</table>

PC = 250  R1 = 400  ACC

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Symbolic Convention</th>
<th>Register Transfer</th>
<th>Effective Address</th>
<th>Contents of ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>LDA ADRS</td>
<td>$ACC \leftarrow M[ADRS]$</td>
<td>500</td>
<td>800</td>
</tr>
<tr>
<td>Immediate</td>
<td>LDA #NBR</td>
<td>$ACC \leftarrow NBR$</td>
<td>251</td>
<td>500</td>
</tr>
<tr>
<td>Indirect</td>
<td>LDA [ADRS]</td>
<td>$ACC \leftarrow M[ADRS]$</td>
<td>800</td>
<td>300</td>
</tr>
<tr>
<td>Relative</td>
<td>LDA $ADRS$</td>
<td>$ACC \leftarrow M[ADRS + PC]$</td>
<td>752</td>
<td>600</td>
</tr>
<tr>
<td>Index</td>
<td>LDA ADRS (R1)</td>
<td>$ACC \leftarrow M[ADRS + R1]$</td>
<td>900</td>
<td>200</td>
</tr>
<tr>
<td>Register</td>
<td>LDA R1</td>
<td>$ACC \leftarrow R1$</td>
<td>—</td>
<td>400</td>
</tr>
<tr>
<td>Register-indirect</td>
<td>LDA (R1)</td>
<td>$ACC \leftarrow M[R1]$</td>
<td>400</td>
<td>700</td>
</tr>
</tbody>
</table>

Refers to Figure 10-6
## Instruction Set Architecture

<table>
<thead>
<tr>
<th></th>
<th>RISC (reduced instruction set computers)</th>
<th>CISC (complex instruction set computers)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory access</strong></td>
<td>restricted to load/store instructions, and data manipulation instructions are register-to-register</td>
<td>is directly available to most types of instructions</td>
</tr>
<tr>
<td><strong>Addressing mode</strong></td>
<td>limited in number</td>
<td>substantial in number</td>
</tr>
<tr>
<td><strong>Instruction formats</strong></td>
<td>all of the same length</td>
<td>of different lengths</td>
</tr>
<tr>
<td><strong>Instructions</strong></td>
<td>perform elementary operations</td>
<td>perform both elementary and complex operations</td>
</tr>
<tr>
<td><strong>Control unit</strong></td>
<td>Hardwired, high throughput and fast execution</td>
<td>Microprogrammed, facilitate compact programs and conserve memory,</td>
</tr>
</tbody>
</table>
### Data Transfer Instructions

- Data transfer: memory\(\leftarrow\rightarrow\) registers, processor registers \(\leftarrow\rightarrow\) input/output registers, among the processor registers

- Data transfer instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>LD</td>
</tr>
<tr>
<td>Store</td>
<td>ST</td>
</tr>
<tr>
<td>Move</td>
<td>MOVE</td>
</tr>
<tr>
<td>Exchange</td>
<td>XCH</td>
</tr>
<tr>
<td>Push</td>
<td>PUSH</td>
</tr>
<tr>
<td>Pop</td>
<td>POP</td>
</tr>
<tr>
<td>Input</td>
<td>IN</td>
</tr>
<tr>
<td>Output</td>
<td>OUT</td>
</tr>
</tbody>
</table>
I/O

- Input and output (I/O) instructions transfer data between processor registers and I/O devices
  - Ports

- Independent I/O system: address range assigned to memory and I/O ports are independent from each other

- Memory-mapped I/O system: assign a subrange of the memory addresses for addressing I/O ports
# Data Manipulation Instructions

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Logical and bit manipulation</th>
<th>Shift instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Mnemonic</strong></td>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Increment</td>
<td>INC</td>
<td>Clear</td>
</tr>
<tr>
<td>Decrement</td>
<td>DEC</td>
<td>Set</td>
</tr>
<tr>
<td>Add</td>
<td>ADD</td>
<td>Complement</td>
</tr>
<tr>
<td>Subtract</td>
<td>SUB</td>
<td>AND</td>
</tr>
<tr>
<td>Multiply</td>
<td>MUL</td>
<td>OR</td>
</tr>
<tr>
<td>Divide</td>
<td>DIV</td>
<td>Exclusive-OR</td>
</tr>
<tr>
<td>Add with carry</td>
<td>ADDC</td>
<td>Clear carry</td>
</tr>
<tr>
<td>Subtract with borrow</td>
<td>SUBB</td>
<td>Set Carry</td>
</tr>
<tr>
<td>Subtract reverse</td>
<td>SUBR</td>
<td>Complement carry</td>
</tr>
<tr>
<td>Negate</td>
<td>NEG</td>
<td></td>
</tr>
</tbody>
</table>

- **Logical and bit manipulation**
  - Clear
  - Set
  - Complement
  - AND
  - OR
  - Exclusive-OR
  - Clear carry
  - Set Carry
  - Complement carry

- **Shift instructions**
  - Logical shift right
  - Logical shift left
  - Arithmetic shift right
  - Arithmetic shift left
  - Rotate right
  - Rotate left
  - Rotate right with carry
  - Rotate left with carry

- **Operands**
  - Immediate (`#NBR`)
  - Register (`R1`)
  - Register indirect (`(R1)`)
  - Direct (`ADR`)
## Typical Shift Instructions

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical shift right</td>
<td>SHR</td>
<td><img src="image1.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Logical shift left</td>
<td>SHL</td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Arithmetic shift right</td>
<td>SHRA</td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Arithmetic shift left</td>
<td>SHLA</td>
<td><img src="image4.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Rotate right</td>
<td>ROR</td>
<td><img src="image5.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Rotate left</td>
<td>ROL</td>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Rotate right with carry</td>
<td>RORC</td>
<td><img src="image7.png" alt="Diagram" /></td>
</tr>
<tr>
<td>Rotate left with carry</td>
<td>ROLC</td>
<td><img src="image8.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Program Control Instructions

- Control over the flow of program execution and a capability of branching to different program segments

- One-address instruction:
  - Jump: direct addressing
  - Branch: relative addressing

<table>
<thead>
<tr>
<th>Name</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch</td>
<td>BR</td>
</tr>
<tr>
<td>Jump</td>
<td>JMP</td>
</tr>
<tr>
<td>Call procedure</td>
<td>CALL</td>
</tr>
<tr>
<td>Return from procedure</td>
<td>RET</td>
</tr>
<tr>
<td>Compare (by subtraction)</td>
<td>CMP</td>
</tr>
<tr>
<td>Test (by ANDing)</td>
<td>TEST</td>
</tr>
</tbody>
</table>
Conditional Branching Instructions

- May or may not cause a transfer of control, depending on the value of stored bits in the *PSR* (processor state register)

<table>
<thead>
<tr>
<th>Branch Condition</th>
<th>Mnemonic</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch if zero</td>
<td>BZ</td>
<td>Z = 1</td>
</tr>
<tr>
<td>Branch if not zero</td>
<td>BNZ</td>
<td>Z = 0</td>
</tr>
<tr>
<td>Branch if carry</td>
<td>BC</td>
<td>C = 1</td>
</tr>
<tr>
<td>Branch if no carry</td>
<td>BNC</td>
<td>C = 0</td>
</tr>
<tr>
<td>Branch if minus</td>
<td>BN</td>
<td>N = 1</td>
</tr>
<tr>
<td>Branch if plus</td>
<td>BNN</td>
<td>N = 0</td>
</tr>
<tr>
<td>Branch if overflow</td>
<td>BV</td>
<td>V = 1</td>
</tr>
<tr>
<td>Branch if no overflow</td>
<td>BNV</td>
<td>V = 0</td>
</tr>
</tbody>
</table>
## Conditional Branching Instructions

### for Unsigned Numbers

<table>
<thead>
<tr>
<th>Branch Condition</th>
<th>Mnemonic</th>
<th>Condition</th>
<th>Status Bits*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch if above</td>
<td>BA</td>
<td>$A &gt; B$</td>
<td>$C + Z = 0$</td>
</tr>
<tr>
<td>Branch if above or equal</td>
<td>BAE</td>
<td>$A \geq B$</td>
<td>$C = 0$</td>
</tr>
<tr>
<td>Branch if below</td>
<td>BB</td>
<td>$A &lt; B$</td>
<td>$C = 1$</td>
</tr>
<tr>
<td>Branch if below or equal</td>
<td>BBE</td>
<td>$A \leq B$</td>
<td>$C + Z = 1$</td>
</tr>
<tr>
<td>Branch if equal</td>
<td>BE</td>
<td>$A = B$</td>
<td>$Z = 1$</td>
</tr>
<tr>
<td>Branch if not equal</td>
<td>BNE</td>
<td>$A \neq B$</td>
<td>$Z = 0$</td>
</tr>
</tbody>
</table>

*Note that $C$ here is a borrow bit.

### for Signed Numbers

<table>
<thead>
<tr>
<th>Branch condition</th>
<th>Mnemonic</th>
<th>Condition</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch if greater</td>
<td>BG</td>
<td>$A &gt; B$</td>
<td>$(N \oplus V) + Z = 0$</td>
</tr>
<tr>
<td>Branch if greater or equal</td>
<td>BGE</td>
<td>$A \geq B$</td>
<td>$N \oplus V = 0$</td>
</tr>
<tr>
<td>Branch if less</td>
<td>BL</td>
<td>$A &lt; B$</td>
<td>$N \oplus V = 1$</td>
</tr>
<tr>
<td>Branch if less or equal</td>
<td>BLE</td>
<td>$A \leq B$</td>
<td>$(N \oplus V) + Z = 1$</td>
</tr>
<tr>
<td>Branch if equal</td>
<td>BE</td>
<td>$A = B$</td>
<td>$Z = 1$</td>
</tr>
<tr>
<td>Branch if not equal</td>
<td>BNE</td>
<td>$A \neq B$</td>
<td>$Z = 0$</td>
</tr>
</tbody>
</table>
Datapath for Branch instruction

**Register file**
- Read register 1
- Read data 1
- Read register 2
- Read data 2
- write register
- write data

**To branch control logic**

**ALU**

**Branch target**

**Zero**

**Sign extend**

**Shift left 2**

**sum**

PC +4 from instruction fetch network

**Conditional Branching Instructions** (Contd.)

<table>
<thead>
<tr>
<th>Branch Condition</th>
<th>Mnemonics</th>
<th>Condition</th>
<th>Status bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch if higher</td>
<td>BH</td>
<td>A &gt; B</td>
<td>C + Z = 0</td>
</tr>
<tr>
<td>Branch if higher or equal</td>
<td>BHE</td>
<td>A •%</td>
<td>C = 0</td>
</tr>
<tr>
<td>Branch if lower</td>
<td>BL</td>
<td>A &lt; B</td>
<td>C = 1</td>
</tr>
<tr>
<td>Branch if lower or equal</td>
<td>BLE</td>
<td>A &quot;%</td>
<td>C + Z = 1</td>
</tr>
<tr>
<td>Branch if equal</td>
<td>BE</td>
<td>A = B</td>
<td>Z = 1</td>
</tr>
<tr>
<td>Branch if not equal</td>
<td>BNE</td>
<td>A ≠%</td>
<td>Z = 0</td>
</tr>
</tbody>
</table>

**Branch if greater**

<table>
<thead>
<tr>
<th>Branch Condition</th>
<th>Mnemonics</th>
<th>Condition</th>
<th>Status bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch if greater or equal</td>
<td>BGE</td>
<td>A •%</td>
<td>N V = 0</td>
</tr>
<tr>
<td>Branch if less</td>
<td>BL</td>
<td>A &gt; B</td>
<td>N V = 1</td>
</tr>
<tr>
<td>Branch if less or equal</td>
<td>BLE</td>
<td>A &quot;%</td>
<td>(N V) + Z = 1</td>
</tr>
</tbody>
</table>

**Procedure Call and Return Instructions**

- **Procedure**: self-contained sequence of instructions that performs a given computational task
- **Call procedure instruction**: one address field
  - Stores the value of the PC (return address) in a temporary location
  - The address in the call procedure instruction is loaded into the PC
- **Final instruction in every procedure**: return instruction
  - Take the return address and load into the PC
- **Temporary Location**: fixed memory location, processor register or memory stack
  - E.g. stack
  - **Procedure call**: SP Å SP - 1; M[SP] Å PC + 4; PC Å Effective address
  - **Return**: PC Å M[SP]; SP Å SP + 1

**Procedure Calls**

- **call** fun1
- **return**

**Program Interrupt**

- Handle a variety of situations that require a departure from the normal program sequence to another service program, similar to a call procedure
- Different from procedure calls:
  - Initiated at an unpredictable point in the program, rather than the execution of an instruction
  - Address of the interrupt service is determined by a hardware procedure
  - The information that defines all or part of the contents of the register set, rather than only the PC, should be stored temporarily
- After finishing interruption, resume to the same state before the interruption
  - **PSR**: other than condition codes, also contains what interrupts allowed, user/system mode indication, etc.

**Type of Interrupts**

- **Hardware interrupts**
  - **External interrupts**:
    - input/output devices requesting transfer of data
    - timing devices time-out event
    - circuit monitoring the power supply detect an impending power failure, in the ISP transfers the register set contents to nondestructive storage like disk, etc.
    - any other external source
  - **Internal interrupts (traps)**:
    - Invalid or erroneous use of an instruction
    - Arithmetic overflow, attempt to divide by zero, an invalid opcode, memory stack overflow, protection violation
- **Software interrupts**: initiated by executing an instruction
  - System call instructions, change from user mode to system mode
Procedure Call and Return Instructions

- **Procedure**: self-contained sequence of instructions that performs a given computational task

- **Call procedure instruction**: one-address field
  - Stores the value of the PC (return address) in a temporary location
  - The address in the call procedure instruction is loaded into the PC

- **Final instruction in every procedure**: return instruction
  - Take the return address and load into the PC

- **Temporary Location**: fixed memory location, processor register or memory stack
  - E.g. stack
    - Procedure call: $SP \leftarrow SP-1; M[SP] \leftarrow PC+4; PC \leftarrow \text{Effective address}$
    - Return: $PC \leftarrow M[SP]; SP \leftarrow SP+1$
Interrupts

Types of Interrupts

1. **External**: Hard Drive, Mouse, Keyboard, Modem, Printer

2. **Internal**: Overflow; Divide by zero; Invalid opcode; Memory stack overflow; Protection violation

3. **Software**: A software interrupt provides a way to call the interrupt routines normally associated with external or internal interrupts by inserting an instruction into the code.
Floating-Point Computation

- What can be represented in N bits?
  - Unsigned: \(0\) to \(2^{N-1}\)
  - 2s Complement: \(-2^{N-1}\) to \(2^{N-1} - 1\)
  - 1s Complement: \(-2^{N-1} + 1\) to \(2^{N-1} - 1\)
  - BCD: \(0\) to \(10^{N/4} - 1\)

- But, what about?
  - very large numbers?
    - \(9,349,398,989,787,762,244,859,087,678\)
  - very small number?
    - \(0.0000000000000000000000000000000045691\)
  - rationals: \(2/3\)
  - irrationals: \(\sqrt{2}\)
  - transcendentals: \(e\)
Recall Scientific Notation

**Issues:**

- Representation, Normal form
- Range and Precision
- Arithmetic (+, -, *, /)
- Rounding
- Exceptions (e.g., divide by zero, overflow, underflow)
- Errors
- Properties (negation, inversion, if \( A \neq B \) then \( A - B \neq 0 \))
Floating-Point Numbers

- **Representation of floating point numbers in IEEE 754 standard:**

  - **Single precision**
    - **Sign:** 1 bit
    - **Exponent:** 8 bits (Biased exponent: actual exponent is $e = E - 127$ for $0 < E < 255$)
    - **Mantissa:** 23 bits (Sign + magnitude, normalized binary significand w/ hidden integer bit: $1.M$)

  - **IEEE F.P.**
    - $1.M \times 2^{e-127}$

- **Exponent field ($E$):**
  - $E=0$ reserved for zero (with fraction $M=0$), and denormalized #s ($M \neq 0$)
  - $E=255$ reserved for $\pm \infty$ (with fraction $M=0$), and NaN ($M \neq 0$)

- **Magnitude of numbers that can be represented is in the range:** (with $E$ in $[1, 254]$):
  
  $2^{-126} (1.8 \times 10^{-38}) \sim 2^{127} (2-2^{-23}) (3.4 \times 10^{38})$
Basic Addition Algorithm

- Steps for addition (or subtraction):
  1. compute $Y_e - X_e$ (**getting ready to align binary point**). $Y_e > X_e$
  2. right shift $X_m$ that many positions to form $X_m \cdot 2^{X_e - Y_e}$
  3. compute $X_m \cdot 2^{X_e - Y_e} + Y_m$

Example: $0.5372400 \times 10^2$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-0.1580000 \times 10^{-1}$</td>
<td>$-0.0001580 \times 10^2$</td>
</tr>
<tr>
<td></td>
<td>$0.5370820 \times 10^2$</td>
</tr>
</tbody>
</table>

if result demands normalization, then normalization step follows:

4. left shift result, decrement result exponent (e.g., 0.001xx...) right shift result, increment result exponent (e.g., 101.1xx...)
   continue until MSB of data is 1  (NOTE: Hidden bit in IEEE Standard)

5. if result is 0 mantissa, may need to zero exponent by special step
### Example

- **Adding operation on two IEEE single precision floating point numbers (X and Y)**

  \[X = 0100 0000 \text{1010 0000 0000 0000 0000 0000} \]
  \[Y = 1100 0000 \text{0011 0000 0000 0000 0000 0000} \]

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>23</td>
</tr>
</tbody>
</table>

\[N = (-1)^S 2^{E-127} (1.M)\]

\[X = (-1)^0 2^{129-127} (1.01) = 2^2 \times 1.01\]
\[Y = (-1)^1 2^{128-127} (1.011) = -2 \times 1.011\]
\[Xe > Ye\]
\[Y = -2^2 \times (1.011 \times 2^{-1}) = -2^2 \times (0.1011)\]
\[X + Y = 2^2 \times (1.01 - 0.1011) = 2^2 \times (0.1001) = 2 \times 1.001\]
\[= 0100 0000 0001 0000 0000 0000 0000 0000\]