CPU Structure

CPU must:
- Fetch instructions
- Interpret instructions
- Fetch data
- Process data
- Write data
Registers

- CPU must have some working space (temporary storage)
- Called registers
- Number and function vary between processor designs
- One of the major design decisions
- Top level of memory hierarchy

User Visible Registers

- General Purpose
- Data
- Address
- Condition Codes
General Purpose Registers (1)

- May be true general purpose
- May be restricted
- Orthogonal: If any general-purpose register can contain the operand for any opcode
- May be used for data or addressing
- Data
  - Accumulator
- Addressing
  - Segment

General Purpose Registers (2)

- Make them general purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Make them specialized
  - Smaller (faster) instructions
  - Less flexibility
How Many GP Registers?

- Between 8 - 32
- Fewer = more memory references
- More does not reduce memory references and takes up processor real estate
- See also RISC

How big?

- Large enough to hold full address
- Large enough to hold full word
- Often possible to combine two data registers
  - C programming
  - `double int a;`
  - `long int a;`
**Condition Code Registers**

- Sets of individual bits
  - e.g. result of last operation was zero
- Can be read (implicitly) by programs
  - e.g. Jump if zero
- Can not (usually) be set by programs

**Control & Status Registers**

- Program Counter
- Instruction Decoding Register
- Memory Address Register
- Memory Buffer Register

Quick quiz: what do these all do?
**Program Status Word (PSW)**

- A set of bits
- Includes Condition Codes
  - Sign of last result
  - Zero
  - Carry
  - Equal
  - Overflow
  - Interrupt enable/disable
  - Supervisor

**Supervisor Mode**

- Intel ring zero
- Kernel mode
- Allows privileged instructions to execute
- Used by operating system
- Not available to user programs
Other Registers

- May have registers pointing to:
  - Process control blocks (see O/S)
  - Interrupt Vectors (see O/S)

- CPU design and operating system design are closely linked, can tailor register organization to the OS
- Often the first few hundred or thousand words of memory allocated for control purposes

### Example Register Organizations

<table>
<thead>
<tr>
<th>Data Registers</th>
<th>General Registers</th>
<th>General Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>AX - Accumulator</td>
<td>EAX - AX</td>
</tr>
<tr>
<td>D1</td>
<td>BX - Base</td>
<td>EBX - BX</td>
</tr>
<tr>
<td>D2</td>
<td>CX - Count</td>
<td>ECX - CX</td>
</tr>
<tr>
<td>D3</td>
<td>DX - Dir</td>
<td>EDX - DX</td>
</tr>
<tr>
<td>D4</td>
<td>Pointer &amp; Index</td>
<td>ESP - SP</td>
</tr>
<tr>
<td>D5</td>
<td>SP - Stack Pointer</td>
<td>EBP - BP</td>
</tr>
<tr>
<td>D6</td>
<td>BP - Base Pointer</td>
<td>ESI - SI</td>
</tr>
<tr>
<td>D7</td>
<td>SI - Source Index</td>
<td>EDI - DI</td>
</tr>
<tr>
<td></td>
<td>Source Index</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dest Index</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address Registers</th>
<th>Segment</th>
<th>Program Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>CS - Code</td>
<td>FLAGS Register</td>
</tr>
<tr>
<td>A1</td>
<td>DS - Data</td>
<td>Instruction Pointer</td>
</tr>
<tr>
<td>A2</td>
<td>SS - Stack</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>ES - Extra</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>DS</td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>SS</td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td>ES</td>
<td></td>
</tr>
</tbody>
</table>

- (a) 3860000
- (b) 8086
- (c) 80386 - Pentium II
Motorola MC68000
- Noted for being an orthogonal architecture
- 8 32-bit general purpose data registers
- 8 32-bit address registers
  - Some used as stack pointers, OS
- 32-bit program counter
- 16-bit status register
- Nice clean architecture, no messy segmentation

Intel 8086
- Other extreme from MC68000, lots of specific registers
- 16-bit flags, Instruction Pointer
- General Registers, 16 bits
  - AX - Accumulator, favored in calculations
  - BX - Base, normally holds an address of a variable or func
  - CX - Count, normally used for loops
  - DX - Data, normally used for multiply/divide
Example Microprocessors –
Register Organization 8086

**Segment, 16 bits**
- SS – Stack, base segment of stack in memory
- CS – Code, base location of code
- DS – Data, base location of variable data
- ES – Extra, additional location for memory data

**Index, 16 bits**
- BP – Base Pointer, offset from SS for locating subroutines
- SP – Stack Pointer, offset from SS for top of stack
- SI – Source Index, used for copying data/strings
- DI – Destination Index, used for copy data/strings

**Move to 32 bit registers in 80386 complicated things a bit for backward compatibility**

Instruction Cycle - Indirect Cycle

**Instruction Cycle**
- In the Fetch Portion, there are three ways to handle addressing in the instruction
  - Immediate Addressing – Operand is directly present in the instruction, e.g. ADD 5 = “Add 5 to Acc”
  - Direct Addressing – The operand contains an address with the data, e.g. ADD 100h = “Add (Contents of Mem Location 100)”
    - Downside: Need to fit entire address in the instruction, may limit address space
  - Indirect Addressing – The operand contains an address, and that address contains the address of the data, e.g. Add (100h) = “The data at memory location 100 is an address. Go to the address stored there and get that data and add it to the Acc.”
    - Downside: Requires more memory accesses
    - Upside: Can store a full address at memory location 100
    - Can also do Indirect Addressing with registers
- Indirect Addressing can be thought of as additional instruction subcycle
Instruction Cycle with Indirect

Instruction Cycle State Diagram
Data Flow (Instruction Fetch)

* Depends on CPU design
* In general:

**Fetch**
- PC contains address of next instruction
- Address moved to MAR
- Address placed on address bus
- Control unit requests memory read
- Result placed on data bus, copied to MBR, then to IR
- Meanwhile PC incremented by 1

Data Flow (Data Fetch)

* IR is examined
* If instruction uses immediate addressing
  - Rightmost N bits of MBR available for processing
* If instruction uses direct addressing
  - Send rightmost N bits of MBR to MAR
  - Control unit requests memory read
  - Result (operand at that address) moved to MBR
* If instruction calls for indirect addressing, indirect cycle is performed
  - Right most N bits of MBR transferred to MAR
  - Control unit requests memory read
  - Result (address of operand) moved to MBR
  - MBR moved to MAR
  - Control unit requests memory read
  - Result (operand at the address) moved to MBR
**Data Flow (Execute)**

- May take many forms
- Depends on instruction being executed
- May include
  - Memory read/write
  - Input/Output
  - Register transfers
  - ALU operations

**Data Flow (Interrupt)**

- Simple
- Predictable
- Repeat the following for all registers that need saving
  - Contents of register copied to MBR
  - Special memory location (e.g. stack pointer) loaded to MAR
  - MBR written to memory
  - Increment stack pointer
- PC loaded with address of interrupt handling routine
- Next instruction (first of interrupt handler) can be fetched
Data Flow (Interrupt Diagram)