CSCI 4717/5717
Computer Architecture

Topic: CPU Registers

Reading: Stallings, Sections 10.3, 10.4, 12.1, and 12.2

CPU Internal Design Issues

- CPU design and operating system design are closely linked
- Compiler also has heavy dependence on CPU design

How Many Instructions are Needed?

Instruction sets have been designed with

- Small numbers of instructions
- Hundreds of instructions
- Trend today is to use “enough” to get the job done well (more on this in the RISC/CISC discussions to come)

How Many Instructions are Needed?

- Until the 1980s, the trend was to construct more and more complex instruction sets containing hundreds of instructions and variations
- Intent was to provide mechanisms to bridge the semantic gap, the difference in high and low level functioning of the computer

Bridging the Semantic Gap

- Reconcile the views of the HLL programmer and the assembly level programmer
- Provide a diverse set of instructions in an attempt to match the programming style of HLL
- Permit the compiler to “bridge the gap” with a single instruction rather than synthesizing a series of instructions
- Did not always have the desired impact

Attributes of a Good Instruction Set

- Wulff asserts that compiler writers might make the better architects because they have had to deal with poor architecture decisions

Wulff's Attributes of a Good Instruction Set

- Complete: be able to construct a machine-level program to evaluate any computable function
- Efficient: frequently performed functions should be done quickly with few instructions
- Regular and complete classes of instructions: provide "logical" set of operations
- Orthogonal: define instructions, data types, and addressing independently
- Additional attribute: Compatible: with existing H/W and S/W in a product line

Addresses in an Instruction

- In a typical arithmetic or logical instruction, 3 references are required
  - 2 operands
  - a result
- These addresses can be explicitly given or implied by the instruction

3 address instructions

- Both operands and the destination for the result are explicitly contained in the instruction word
- Example: X = Y + Z
- With memory speeds (due to caching) approaching the speed of the processor, this gives a high degree of flexibility to the compiler
- Avoid the hassles of keeping items in the register set -- use memory as one large set of registers
- This format is rarely used due to the length of addresses themselves and the resulting length of the instruction words

2 address instructions

- One of the addresses is used to specify both an operand and the result location
- Example: X = X + Y
- Very common in instruction sets
- Supported by heavy use in HLL of operations such as A += B or C <<=3;

```
ADD A,B          ;A = A + B
```


1 address instructions

- When only a single reference is allowed in an instruction, another reference must be included as part of the instruction
- Traditional accumulator-based operations
- Example: Acc = Acc + X
- For an instruction such as A += B, code must first load A into an accumulator, then add B.

```
LOAD A
ADD B
```


0 address instructions

- All addresses are implied, as in register-based operations – e.g., TBA (transfer register B to A)
- Stack-based operations
- All operations are based on the use of a stack in memory to store operands
- Interact with the stack using push and pop operations

Trade off Resulting from Fewer Addresses

Fewer addresses in the instruction results in:

- More primitive instructions
- Less complex CPU
- Instructions with shorter length – fit more into memory
- More total instructions in a program
- Longer, more complex programs
- Faster fetch/execution of instructions
- Longer execution times

Example: 3 Addresses

Y = (A-B) / (C+D*E)

```
SUB Y,A,B
MUL T,D,E
ADD T,T,C
DIV Y,Y,T
```


Example: 2 address

Y = (A-B) / (C+D*E)

```
MOV Y,A
SUB Y,B
MOV T,D
MUL T,E
ADD T,C
DIV Y,T
```


Example: 1 address

Y = (A-B) / (C+D*E)

```
LOAD D
MUL E
ADD C
STORE Y
LOAD A
SUB B
DIV Y
STORE Y
```


Example: 0 address – Convert to postfix (reverse Polish) notation:

Y = (A-B) / (C+D*E)

becomes

Y = AB–CDE*+/

This is "Postfix" or "Reverse Polish Form" from tree searching.

```
PUSH A
PUSH B
SUB
PUSH C
PUSH D
PUSH E
MUL
ADD
DIV
POP Y
```


Design Decisions

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types – various types of operations and how they are performed
- Instruction formats
  - Length of op code field
  - Number of addresses

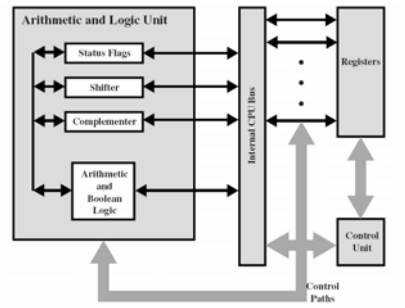
CPU Internal Design Issues

From our discussion of the architecture of the computer, we've put some requirements on the CPU:

- Fetch instructions from memory
- Interpret instructions to determine action that is required
- Fetch data that may be required for execution (could come from memory or I/O)
- Process data with arithmetic, logic, or some movement of data
- Write data to memory or I/O

CPU Internal Structure

Design decisions here affect instruction set design



CPU Internal Structure (continued)

- Arithmetic Logic Unit
  - Status flags
  - Shifter
  - Complementer
  - Arithmetic logic
  - Boolean logic
- Internal CPU bus to pass data back and forth between items of CPU

CPU Internal Structure (continued)

Registers

- CPU must have some working space (temporary storage) to remember things
  - Data
  - location of last instruction or next instruction
  - instruction as it's working with it
- Number and function vary between processor designs
- One of the major design decisions
- Absolute top level of memory hierarchy

CPU Internal Structure (continued)

Two types of registers:

- User-visible registers -- allow for operations with minimal interaction with main memory (programmer takes place of cache controller)
- Control and Status Registers -- with correct privileges, can be set by programmer. Lesser privileges may provide read-only capability.

CPU Internal Structure (continued)

- Control unit -- managing operation of all CPU items
- Internal CPU bus to pass data back and forth between items of CPU

User Visible Registers

- Accessed through machine/assembly language instructions
  - General Purpose
  - Data
  - Address
  - Condition Codes
- Represent complete user-oriented view of processor -- therefore, storing and later restoring of all user-visible registers effectively resets processor back to stored state

General Purpose Registers

- May be true general purpose -- can contain the operand for any opcode
- May be restricted -- floating point only, integer only, address only
- May be used for data or addressing -- some may do either address or data, in some cases there may be a clear distinction between data and address registers
- Accumulator → Data
- Addressing
  - Segment
  - Index -- may be autoindexed
  - Stack

Register Design Issues

The range of design decisions goes from…

- Make all registers general purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Make them specialized
  - Smaller more specialized (faster) instructions
  - Less flexibility

Register Design Issues (continued)

How many general purpose registers?

- Number affects instruction set design => more registers means more operand identifier bits
- Between 8 – 32
- Remember that the registers are at top of hierarchy → faster than cache
- The fewer GP registers, the more memory references
- More does not necessarily reduce memory references and takes up processor real estate
- RISC needs are different and will be discussed later

Register Design Issues (continued)

How big do we make the registers?

- Address -- large enough to hold full address
- Data -- large enough to hold full word
- Often possible to combine two data registers -- e.g. AH + AL = AX
- Example: Do we link the design of registers to a standard, e.g., C programming
  - double int a;
  - long int a;

Condition Code Registers (flags)

- Sets of individual bits each with a unique purpose (e.g. result of last operation was zero)
- Opcodes can read flag values to determine effect/operation (e.g., conditional jumps)
- Automatically set as a result of some operations
- Some processors allow user to set or clear them explicitly
- Collected into group and referred to as a single register (CCR)

Control & Status Registers

Types of control & status registers

- Registers for movement of data between CPU and memory
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR)
  - Memory Buffer Register (MBR)
- Optional buffers used to exchange data between ALU, MBR, and user-visible registers
- Program Status Word (PSW)
- Address pointers used for control
- Built-in processor I/O control & status registers

Control & Status Registers (continued)

- Program Counter (PC)
  - Automatically incremented to next instruction as part of operation of current instruction
  - Can also be changed as result of jump instruction
- Instruction Register (IR)
  - Most recently fetched instructions
  - Where instruction decoder examines opcode to figure out what to do next

Control & Status Registers (continued)

- Memory Address Register (MAR)
  - Memory address of current memory location to fetch
  - Could be instruction or data
- Memory Buffer Register (MBR)
  - Last word read from memory (instruction or data)
  - Word to be stored to memory

Control & Status Registers (continued)

Program Status Word (PSW) – May be exactly the same thing as user-visible condition code register

- A set of bits which include condition codes
  - Sign of last result
  - Zero
  - Carry
  - Equal
  - Overflow
  - Interrupt enable/disable
  - Supervisor
    - Examples: Intel ring zero, kernel mode
    - Allows privileged instructions to execute
    - Used by operating system
    - Not available to user programs

Control & Status Registers (continued)

- Address pointers used for control
  - Interrupt vectors
  - System stack pointer
  - Page table pointer for hardware supported virtual memory
  - Chip select controls
- On processor I/O
  - Status and control to operate the I/O
  - E.g., serial ports -- bps rate, interrupt enables, buffer registers, etc.