EEC170
Computer Architecture

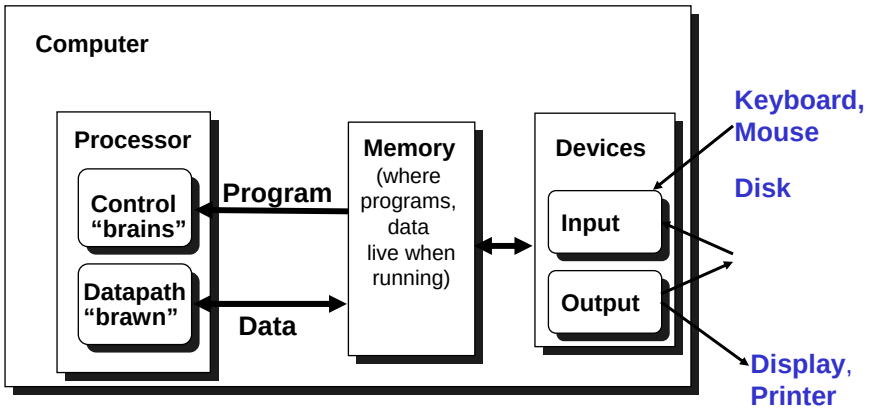
Lecture 2: Instruction Set Architectures

October 10, 2005

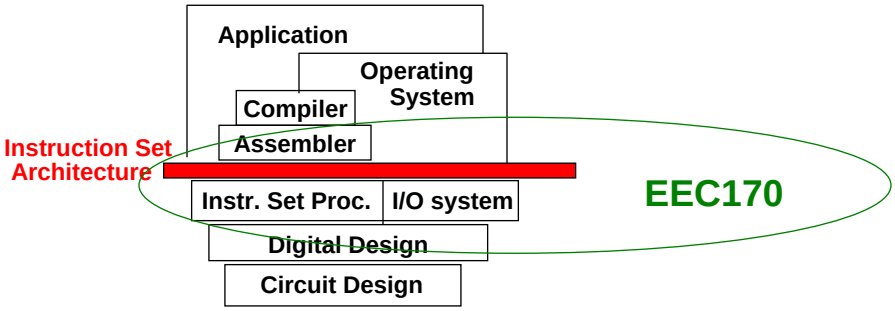
Soheil Ghiasi
Electrical and Computer Engineering
University of California, Davis

Review: What is a Computer?

- **It has storage (data, programs)**
  - **It has a language. Programs are expressed in that language**
- **It has a Processor (Datapath + Control unit)**
- **It has an input/output mechanism**

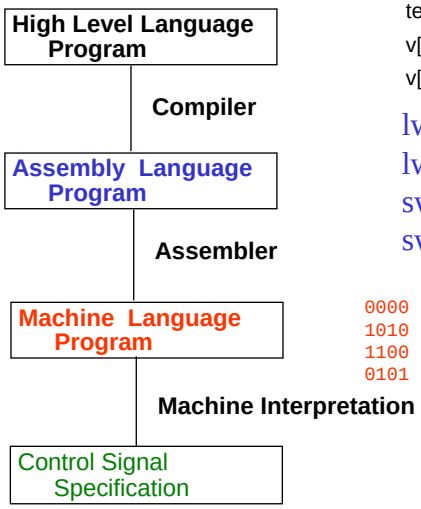


Review: What is "Computer Architecture"



° Co-ordination of *levels of abstraction*

Review: Levels of Representation




```
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
```

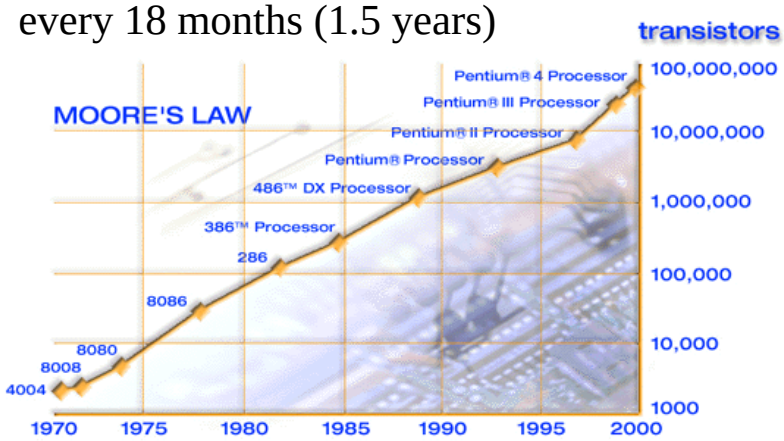
ALUOP[0:3] <= InstReg{9:12] & MASK

Computer Technology – Dramatic Change!

- Processor
  - 2X in speed every 1.5 years (since ‘85); 100X performance in last decade.
- Memory
  - DRAM capacity: 2x / 2 years (since ‘96); 64x size improvement in last decade.
- Disk
  - Capacity: 2X / 1 year (since ‘97)
  - 250X size in last decade.

Review: Moore’s Law

- In 1965, Gordon Moore predicted that the number of transistors per chip would double every 18 months (1.5 years)



Types of Machine Organization

- Memory-to-Memory Machines
- Accumulator
- Stack
- General Purpose Register Machines

Memory to Memory Machine

- **Every instruction contains a full memory address for each operand**
- **Assumptions**
  - **two operands per operation**
  - **second operand is also the destination**
  - **memory address 16 bits (2 bytes)**
  - **operand size 32 bits (4 bytes)**
  - **instruction code 8 bits (1 byte)**
  - **we want to evaluate A ← (B+C+D+E)/X**

Memory to Memory Machines

A ← (B+C+D+E)/X

- Hypothetical assembly language code
  - move B, A; A ← B
  - add C, A; A ← A + C (B+C)
  - add D, A; A ← A + D (B+C+D)
  - add E, A; A ← A + E (B+C+D+E)
  - div X, A; A ← A / X
- An add or divide instruction results in the transfer of 17 bytes between memory and CPU
  - 5 bytes for instruction (opcode + 2 memory addresses)
  - 4 bytes each to fetch 1st and 2nd operands (8 bytes total)
  - 4 bytes to store result
- Move requires 13; Total memory traffic = 4*17 + 13 = 81 bytes

How many bytes transferred between memory + CPU for Add or Divide?

How many bytes transferred between memory + CPU for move?

Why use CPU Storage?

- Consider the idea of providing a small amount of storage in the CPU
  - Goal To reduce memory traffic by keeping repeatedly used operands in the CPU and thus
    - Avoid re-referencing memory
    - Avoid having to specify full memory address of the operand
- This is a perfect example of optimizing the frequent case (Amdahl's Law)

Accumulator Machines

- CPU storage consists of a single accumulator
- Accumulator is an (implicit) operand for most instructions
- Memory is source of 2nd operand for 2 operand instructions
- Cheap (in terms of CPU hardware) and simple
- Memory traffic can be reduced significantly more if more local storage is available

Accumulator Machine

- Consider a machine with 1 cell of CPU storage: the accumulator
- Assumptions
  - two operands per operation
  - 1st operand is in the accumulator (implicit)
  - 2nd operand is in memory
  - accumulator is also the destination of the operation except for store
  - memory address 16 bits (2 bytes)
  - operand size 32 bits (4 bytes)
  - instruction code 8 bits (1 byte)
- we want to evaluate A ←(B + C + D + E) / X

Accumulator Machine

- Assembly language code hypothetical
  - load B; acc ← B
  - add C; acc ← acc + C (B+C)
  - add D; acc ← acc + D (B+C+D)
  - add E; acc ← acc + E (B+C+D+E)
  - div X; acc ← acc / X
  - store A; A ← acc
- Each of the above instructions results in the transfer of 7 bytes between memory and CPU
  - 3 bytes for instruction 4 bytes to fetch or store 2nd operand
  - Total memory traffic 6x7 = 42 bytes
- Moral: local CPU storage reduces memory traffic and also effects the instruction set design

How many bytes transferred between memory + CPU for load2 add2 div2 store2

Stack Machine

- Most instructions manipulate the top few data items (mostly top 2) of a pushdown stack
- Additional instructions are provided to move data between memory and top of stack
- Top few items of the stack are kept in the CPU
- Instructions manipulate the top of the stack implicitly
- Ideal for evaluating expressions (stack holds intermediate results)
- Were thought to be a good match for high level languages
- Awkward:
  - become very slow if stack grows beyond CPU local storage
  - no simple way to get data from middle of stack

Stack Machines

- **Binary arithmetic and logic operations**
  - **operands: top 2 items on stack**
  - **operands are removed from stack**
  - **result is placed on top of stack**
- **Unary arithmetic and logic operations**
  - **operand: top item on the stack**
  - **operand is replaced by result of operation**
- **Data move operations**
  - **push: place memory data on top of stack**
  - **pop: move top of stack to memory**

Stack Machines: Example Program

- **Evaluate expression A ← (B + C + D + E) / X**

push B;	**top of stack "tos":**	**B**
push C;	**tos:**	**B, C**
add;	**tos:**	**B+C**
push D;	**tos:**	**B+C, D**
add;	**tos:**	**B+C+D**
push E	**tos:**	**B+C+D, E**
add;	**tos:**	**B+C+D+E**
push X;	**tos:**	**B+C+D+E, X**
div;	**tos:**	**B+C+D+E/X**
pop a;	**A ← (B+C+D+E)/X**	

- **An add/divide instruction results in the transfer of 17 bytes between memory and CPU**
  - **1 bytes for instruction opcode**
  - **0 bytes for data transfer**
  - **Push/pop requires 7**
- **Total memory traffic = 6*7 + 4*1 = 46 bytes**

How many bytes transferred between memory + CPU for Add? Divide?

How many bytes transferred between memory + CPU for move?

General Purpose Register Machine

- With stack machines only the top two elements of the stack are directly available to instructions. In general purpose register machines the CPU storage is organized as a set of registers which are equally available to the instructions
- Frequently used operands are placed in registers (under program control)
  - Reduces instruction size
  - Reduces memory traffic

GPR Machine: Example Program

- Evaluate A ←(B + C + D + E) / X on a hypothetical machine with 16 registers R0 to R15 and 2 operand register-register ALU operations
  - load B, R1; R1 ← B
  - load C, R2; R2 ← C
  - load D, R3; R3 ← D
  - load E, R4; R4 ← E
  - load X, R5; R5 ← X
  - add R1, R2; R1 ← B+C
  - add R1, R3; R1 ← B+C+D
  - add R1, R4; R1 ← B+C+D+E
  - div R1, R5 R1 ← (B+C+D+E)/X
  - store R1, A; A ← (B+C+D+E)/X
- How many bytes for load/store? 7 ½ bytes
- How many bytes for add/div? 2 bytes
- Total mem. Xfer = 6*7 ½ + 4 *2 = 53 bytes

Classifying GPR Machine

- GPR machines are subclassifed based on whether or not memory operands can be used by typical ALU instructions
- Register-memory machines: machines where some ALU instructions can specify at least one memory operand and one register operand
- Load-store machines: register-register machines with the restriction that the only instructions that can access memory are the load and the store instructions
  - load transfers data from memory to a register
  - store transfers data from a register to memory

Classification of Instruction Set Architectures

- Classification of stored programs computers based on CPU storage organization
- Characterize instructions and machines by the number of explicit memory addresses of operands in its instructions excluding data move instructions
  - Stack machines: 0 address machines
  - Accumulator machines: 1 address machines
  - General purpose register machines 1, 2, 3 (or more) address machines

Recap: Basic ISA Classes

- Memory to Memory:
  - 2 address — add A B — mem[A] ← mem[A] + mem[B]
  - 3 address — add A B C — mem[A] ← mem[B] + mem[C]
- Accumulator:
  - 1 address — add A — acc ← acc + mem[A]
  - 1+x address — addx A — acc ← acc + mem[A + x]
- Stack:
  - 0 address — add — tos ← tos + next
- General Purpose Register:
  - 2 address — add A B — reg[A] ← reg[A] + reg[B]
  - 3 address — add A B C — reg[A] ← reg[B] + reg[C]

Comparison:

Bytes per instruction? Number of Instructions? Cycles per instruction?

Instruction Classes, Format, Addressing Modes

- Instruction Classes
  - Expect new instruction set architectures to use general purpose register
- Instruction Format
  - If code size is most important, use variable length instructions
  - If performance is most important, use fixed length instructions
- Data Addressing Modes
  - Frequent: Displacement, Immediate, Register Indirect
  - Displacement size should be 12 to 16 bits
  - Immediate size should be 8 to 16 bits
- Operand Sizes
  - Support these data sizes and types:
    8-bit, 16-bit, 32-bit, 64-bit integers and
    32-bit and 64-bit IEEE 754 floating point numbers

Conclusion

- Instruction Set Architecture is the key abstraction between hardware designer and software developers
- Machine Organizations
  - Memory-to-Memory Machines
  - Accumulator
  - Stack
  - General Purpose Register Machines