COMPUTER ORGANIZATION (IS F242)

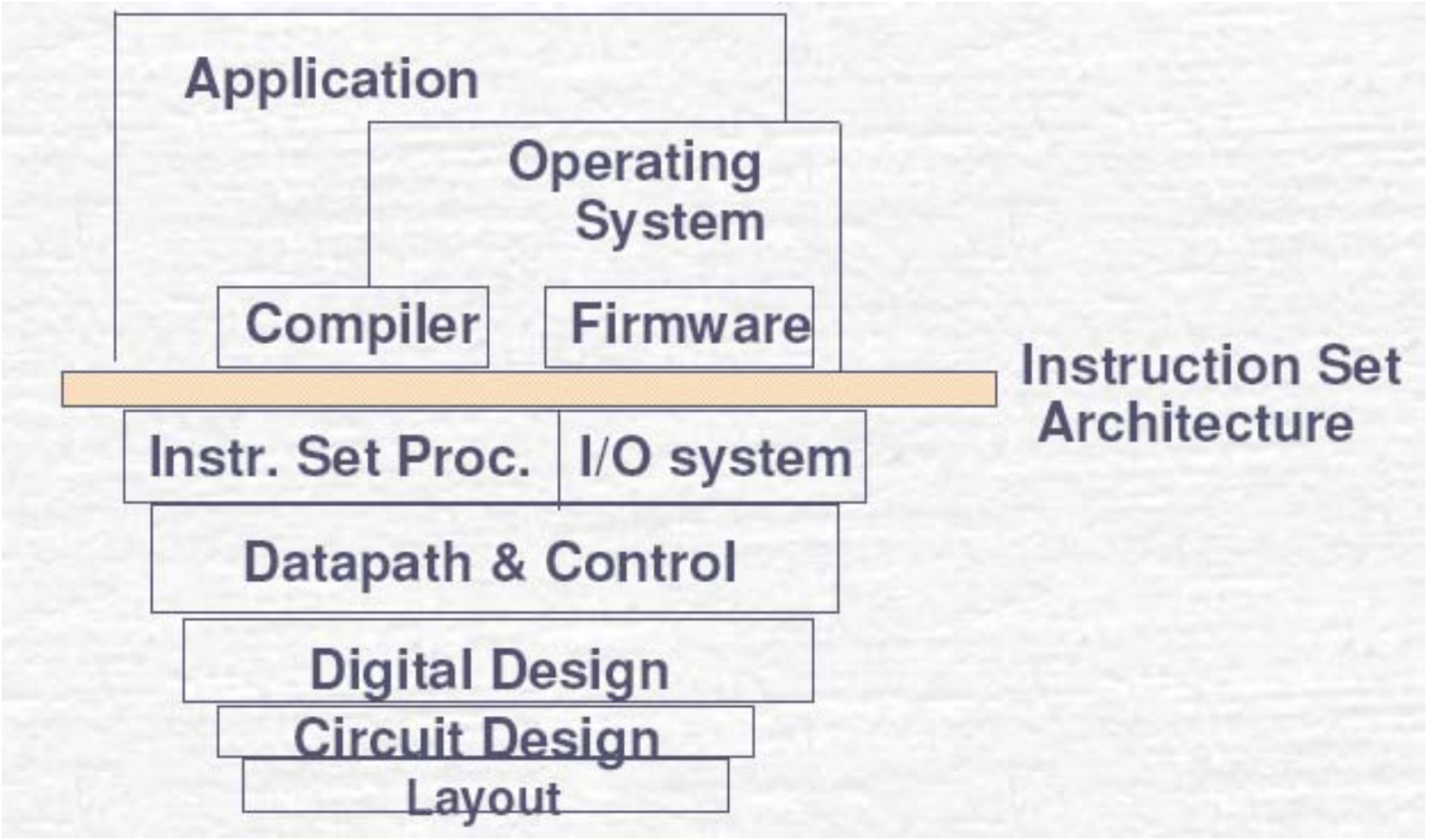
LECT 04_05: COMPUTER ORGANIZATION

Architecture & Organization

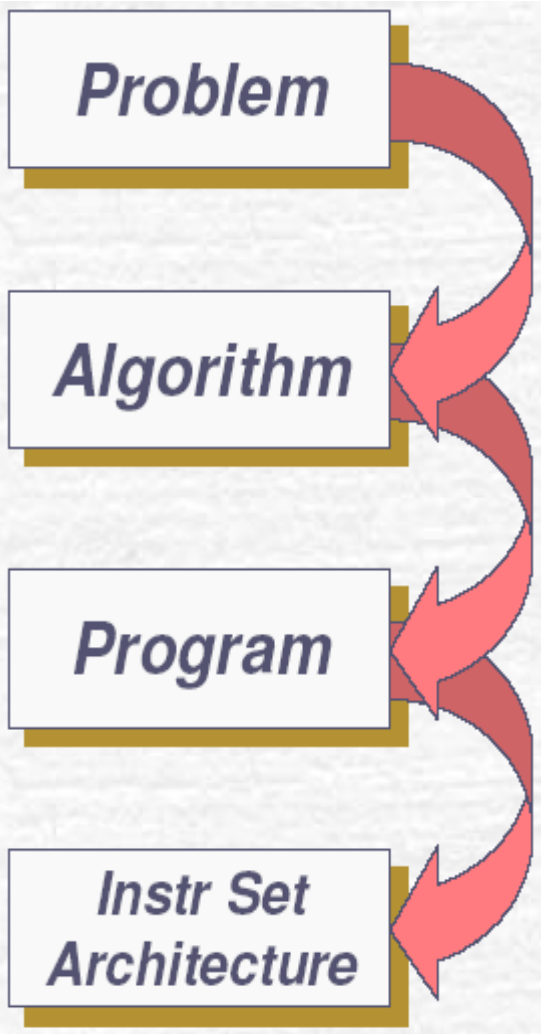
- All Intel x86 family share the same basic architecture
- The IBM System/370 family share the same basic architecture
- This gives code compatibility
  - At least backwards
- Organization differs between different versions

What is Computer Architecture?

Coordination at many levels of abstraction



Transformations Between Layers



Software Design:

Choose Data Structures and Algorithms

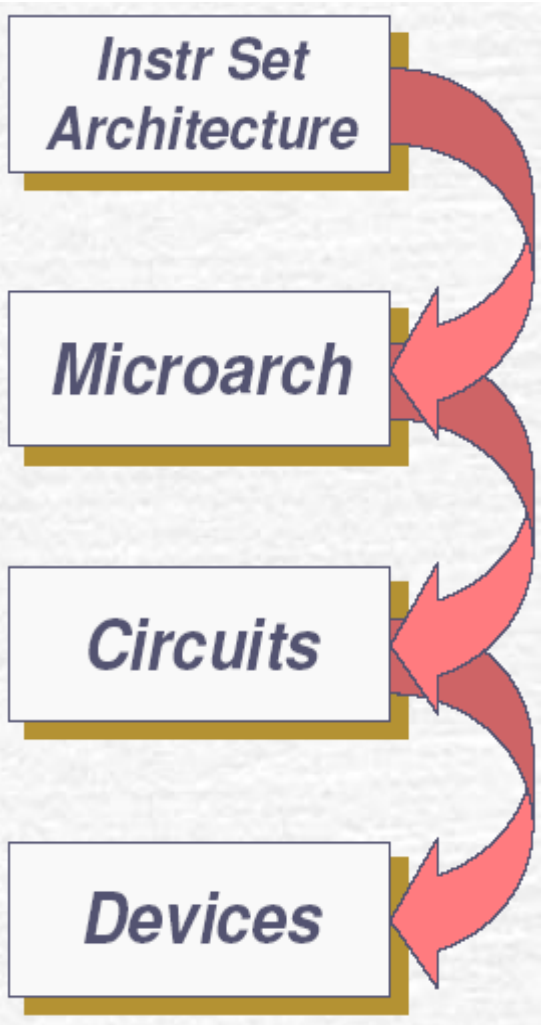
Programming:

Use Language to express Design

Compiling/Interpreting:

Convert Language to Machine Instructions

Transformations Between Layers



Processor Design:

Choose structures to implement ISA

Logic Circuit Design:

Gates and low level circuits to implement components

Process Engineering & Fabrication:

Develop and manufacture lowest level components

Architecture Vs Organisation

Programmer's View

- Organisation of Programmable storage
- Data types, Data structures: Encoding & representations
- Instruction set
- Instruction formats
- Modes of addressing & accessing data items & instructions
- Exceptional conditions

Logic Designer's View

- Functional Units (Registers, ALU, Shifters, Logic Units, ….)
- Ways in which these components are interconnected
- Information flows between components
- Logic and means by which such information flow is controlled
- Organizing FUs to realize the ISA
- Register Transfer Logic (RTL) description

Microprocessors

- 8086,8088,80286 contain 16 bit internal architecture
- 80386-Pentium 4 contain 32 bit internal architecture
- Pentium 4 and Core – 2 are available with 64 bit internal architecture
- Program Visible registers
  - Multipurpose and Special purpose registers
- Invisible registers
  - Addressable in system programming
  - Only 80286 and above contains

Microprocessor Registers

- Program Visible registers
  - Addressable during application programming
  - **RAX, EAX, AX, AH, or AL**
  - **RBX, EBX, BX, BH, or BL**
  - **RCX, ECX, CX, CH, or CL**
  - **RDX, EDX, DX, DH, or DL**
  - **RBP, EBP, BP**
  - **RSI, ESI, SI**
  - **RDI, EDI, DI**
  - **RSP, ESP, SP**
  - **RFLAGS, EFLAGS, FLAGS**
  - **RIP, EIP, IP**
  - **CS, DS, ES, SS, FS, GS**
  - **R8 TO R15**
  - **RAX,RBX,RCX,RDX,RBP,RSI & RDI are for multi purpose**

Multipurpose Registers

- RAX – Accumulator
  - Addressable as RAX, EAX, AX, AH, or AL
  - Mainly used for multiplication & division
  - Sometimes hold the offset address of a location in the memory system (in 80386 and above)
- RBX – Base Index
  - Sometimes hold the offset address of a location in the memory system (in all versions of the microprocessor)
  - Can address memory data (in 80386 and above)
- RCX – Count
  - Holds count for instructions (repeated string instructions – REP/REPE/REPNE, shift, rotate and LOOP/LOOPD)
  - Sometimes hold the offset address of memory data (in 80386 and above)

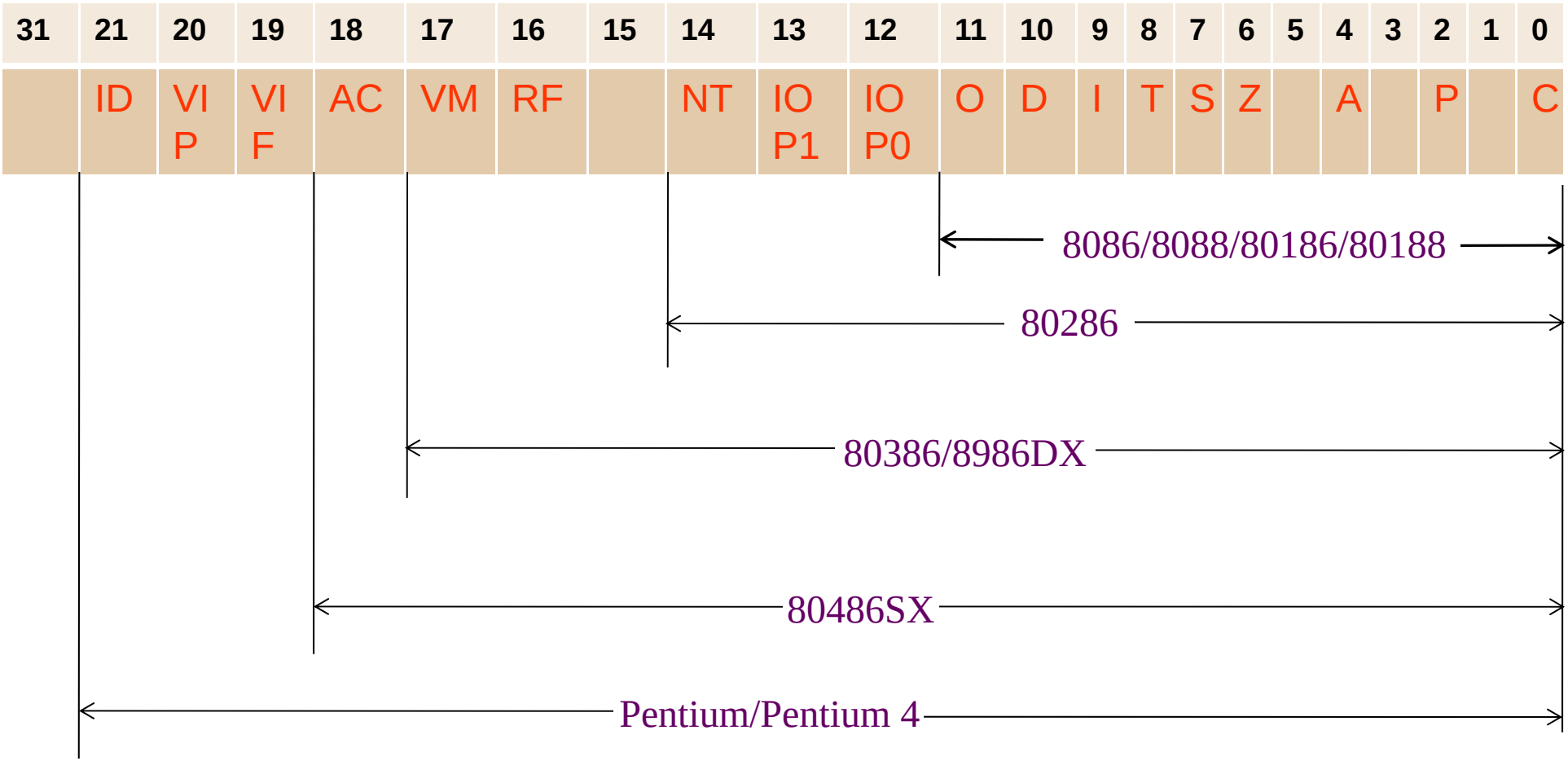
Multipurpose Registers

- RDX – Data
  - Holds a part of the result from a multiplication or part of the dividend before a division
  - Can address memory data (in 80386 and above)
- RBP – Base Pointer
  - Points to a memory location for memory data transfer
- RDI – Destination Index
  - Addresses string destination data for the string instructions
- RSI – Source Index
  - Often addresses source string data for the string instruction
- R8 to R15 – Only in Pentium 4 & Core 2 (64 bit)
  - Bit 8 to 15 are not directly addressable as a byte

Special Purpose Registers

- RIP- Instruction Pointer
  - Addresses the next instruction in a section of memory defined as a code segment
  - Addresses IP in real mode and EIP (80386 and above) in protected mode
  - Can be modified with a jump or a call instruction
- RSP- Stack Pointer
  - Addresses an area of memory called the stack
  - Addresses as SP or ESP
- RFLAGS
  - For controlling the microprocessor operation
  - Addresses as FLAG or EFLAG
  - No change for data transfer or program control operation

RFLAGS


31 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ID VIP VIF AC VM RF NT IOP1 IOP0 O D I T S Z A P C
8086/8088/80186/80188
80286
80386/8986DX
80486SX
Pentium/Pentium 4

RFLAGS

- C (Carry)
  - Holds the carry after addition or borrow after subtraction
- P (Parity)
  - 0 for odd parity and 1 for even parity (count of ones)
- A (Auxiliary Carry)
  - Holds the half carry after addition or borrow after subtraction
- Z (Zero)
  - Result of arithmetic or logic operation is zero?
- S (Sign)
  - Result of arithmetic or logic operation is positive? (S=0 positive)
- T (Trap)
  - Enables trapping through an on-chip debugging feature (if 1)

RFLAGS

- I (Interrupt)
  - Controls operation of INTR input pin (If 1 INTR pin is enabled)
  - State of I flag bit is controlled by STI & CLI (set & clear I flag)
- D (Direction)
  - Selects either the increase or decrease mode for the DI and/or SI registers during string operations (D=1, decrement)
- O (Overflow)
  - Occurs when signed numbers are added or subtracted
- IOPL (I/O privilege level)
  - Used in protection mode to select the privilege level for I/O devices. I/O executes only if the current privilege level is higher / more trusted than the IOPL.
  - IOPL value of 00 is the highest or most trusted.

RFLAGS

- NT (nested task)
  - Current task is nested with in another task in protected mode operation. Flag is set when the task is nested by software.
- RF (resume)
  - Used with debugging to control the resumption of execution after the next instruction
- VM (virtual mode)
  - Selects virtual mode operation in a protected mode system
  - Allows multiple 1M DOS memory partition to coexist
  - Used to simulate DOS in the modern WINDOWS environment
- AC (alignment check)
  - Activates if a word or double word is addressed on a non-word or a non-double word boundary

RFLAGS

- VIF (virtual interrupt)
  - Is a copy of Interrupt flag bit available to the Pentium – Pentium 4 microprocessors
- VIP (virtual interrupt pending)
  - Provides information about virtual mode interrupt for the Pentium – Pentium 4 microprocessors
  - Used in multi-tasking environment to provide OS with virtual interrupt flag and interrupt pending information
- ID (identification)
  - Indicates Pentium – Pentium 4 microprocessors support the CPUID instruction.
  - CPUID instruction provides the system with information about pentium processor (version number & manufacturer)

Segment Registers

- Generate memory addresses when combined with other registers in the processor
- Functions differ in real mode and protected mode
- CS-Code Segment
  - Contains programs and procedures used by the microprocessor
  - Defines the starting address of the section of memory holding code
  - In real mode , it defines the start of a 64KB section of memory
  - In protected mode, it selects a descriptor that describes the starting address and length of a section of memory holding code
- DS- Data Segment
  - Contains data used by the program
  - Data are accessed by an offset address or the contents of other registers that holds the offset address
  - Length is limited to 64KB in 8086-80286 & 4GB in 80386 onwards

Segment Registers

- ES- Extra Segment
  - An additional data segment
  - Used by string instructions to hold destination data
- SS- Stack Segment
  - Defines the area of memory used for the stack
  - Stack entry point is determined by the stack segment and stack pointer registers
  - BP register also addresses data within the stack segment
- FS and GS
  - Supplement segment registers
  - Available in the 80386 – Core 2 microprocessors (allows 2 additional memory segments for access by programs)
  - Windows uses these segments for their internal operations

Real Mode Memory Addressing

- 8086 and 8088 operate exclusively in the real mode
  - 80286 and above operate either in real or protected mode
  - No real mode operation in 64 bit Pentium 4 and Core 2
- Allows the processor to address only the first 1Mb of memory space called as Real memory or Conventional memory or DOS memory
- DOS OS requires the processor to operate in real mode (Windows does not use real mode)
- Real mode helps application written in 8086/8088 to run in 80286 and above

Segments and Offsets in Real Mode

- Memory Address = Segment Address + Offset Address
- One of the segment registers contains segment address
  - i.e. beginning address of any 64KB memory segment
- Offset Address – Any location within 64KB
- Ex. Segment register contains 1000H then what is the starting and ending address of the segment?
  - Starting Address = ???
  - 10000 (0 appended in real mode)
  - Ending Address = ???
  - 1FFFF (each segment is 64KB size)

- Default 16 bit segment and offset combinations
  - CS:IP — Instruction Address
  - SS:SP or BP — Stack Address
  - DS:BX,DI,SI, 8/16 bit number — Data Address
  - ES: DI for string operations — String Destination Address
- Default 32 bit segment and offset combinations
  - CS:EIP — Instruction Address
  - SS:ESP or EBP — Stack Address
  - DS:EAX,EBX, ECX, EDX, EDI, ESI, 8/32 bit number — Data Address
  - ES: EDI for string operations — String Destination Address
  - FS: No default — General Address
  - GS: No Default — General Address

Default segment and offset registers

- CS:IP or CS:EIP
  - CS defines start of the code segment and the IP locates the next instruction within the code segment
  - The combination locates the next instruction executed by the μp
- SS:SP / SS:BP or SS:ESP / SS:EBP
  - Stack data are referenced through the SS and memory location addressed either by SP or BP
  - Ex: SS = 3000H and SP=0500H
  - Memory location addressed by μp = 30500H
- DS:BX / DS:DI / DS:SI / DS:23H / DS:0AC2H
  - For addressing data
- ES:DI Used for string instructions

Advantage of segment –offset scheme

- Programs can be ***Relocated*** in the memory system
- Memory structure is different from machine to machine
- Memory segment can be moved to any place in the memory system without changing any of the offset addresses
- Only the contents of the segment registers required to be change
- Windows assumes that first 2G of memory is available for code and data

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
  - Machine Code or Binary
  - Usually represented by assembly codes
  - In machine code each instruction has a unique bit pattern
  - For human consumption (well, programmers anyway) a symbolic representation is used
    - e.g. ADD, SUB, LOAD
  - Operands can also be represented in this way
    - ADD A,B

Number of Addresses

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything
- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - Temporary storage to hold some results

Number of Addresses

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines
  - Ex. ADD C, LOAD D, SUB B
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c
  - c = a + b

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - Instructions with shorter length
  - More instructions per program
  - Faster fetch/execution of instructions
  - Longer program execution time and more complex programs

Design Decisions

- Operation selection
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes (later…)
- CISC Vs RISC

Instruction Length - RISC v CISC

- CISC has variable instruction encoding
  - 8086 ?
  - Pentium ?
  - VAX ?
- RISC has fixed instruction encoding
  - MIPS
  - SPARC
  - ALPHA DEC
  - PPC

Driving force for CISC

- Increasingly complex high level languages
- Semantic gap Leads to:
  - Large instruction sets
  - More addressing modes
- Intension of CISC
  - Ease compiler writing
    - Compiler simplification?
    - Complex machine instructions harder to exploit
    - Optimization is more difficult
  - Improves execution efficiency

RISC–Reduced Instruction Set Computing

- Features
  - Large number of general purpose registers
    - Use compiler to optimize the register use
  - Limited and simple [but powerful] instruction set
  - Emphasis on optimizing the instruction pipeline
- Characteristics
  - One instruction per cycle [high clock speed]
  - Register to register operations
  - Few simple addressing modes & instruction formats
  - Hardwire (no micro code) control unit design
  - Fixed instruction format