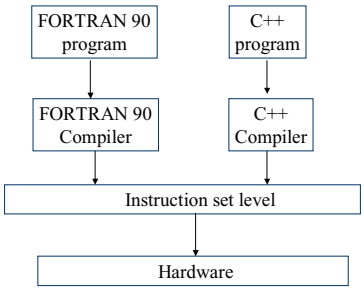
Computer Systems Architecture

Processor Types
And
Instruction Sets

Interfacing Compiler and Hardware



What Instructions Should A Processor Offer?

- Minimum set is sufficient, but inconvenient
- Extremely large set is convenient, but inefficient
- Architect must consider additional factors
  - – Physical size of processor
  - – Expected use
  - – Power consumption

The Point About Instruction Sets

The set of operations a processor provides represents a tradeoff among the cost of the hardware, the convenience for a programmer, and engineering considerations such as power consumption.

Representation

- Architect must choose
  - – Set of instructions
  - – Exact representation hardware uses for each instruction (*instruction format*)
  - – Precise meaning when instruction executed
- Above items define the *instruction set*

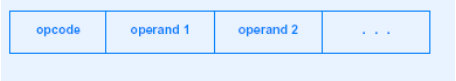
Parts Of An Instruction

- Opcode specifies instruction to be performed
- Operands specify data values on which to operate
- Result location specifies where result will be placed

Instruction Format

- Instruction represented as binary string
- Typically
  - – Opcode at beginning of instruction
  - – Operands follow opcode

Illustration Of Typical Instruction Format



Instruction Length

- Fixed-length
  - – Every instruction is same size
  - – Hardware is less complex
  - – Hardware can run faster
- Variable-length
  - – Some instructions shorter than others
  - – Appeals to programmers
  - – More efficient use of memory

The Point About Fixed-Length Instructions

When a fixed-length instruction set is employed, some instructions contain extra fields that the hardware ignores. The unused fields should be viewed as part of a hardware optimization, not as an indication of a poor design.

General-Purpose Registers

- High-speed storage device
- Typically part of the processor
- Each register small size (typically, each register can accommodate an integer)
- Basic operations are *fetch* and *store*
- Numbered from 0 through N–1
- Many processors require operands for arithmetic operations to be placed in general-purpose registers

Floating Point Registers

- Usually separate from general-purpose registers
- Each holds one floating-point value
- Many processors require operands for floating point operations to be placed in floating point registers

Example Of Programming With Registers

- Add X and Y, and place result in Z
- Steps
  - – Load a copy of X into register 3
  - – Load a copy of Y into register 4
  - – Add the value in register 3 to the value in register 4, and direct the result to register 5
  - – Store a copy of the value in register 5 in Z
- Note: assumes registers 3, 4, and 5 are free

Types Of Instruction Sets

- Two basic forms
  - – Complex Instruction Set Computer (CISC)
  - – Reduced Instruction Set Computer (RISC)

CISC Instruction Set

- Many instructions (often hundreds)
- Given instruction can require arbitrary time to compute
- Examples of CISC instructions
  - – Move graphical item on bitmapped display
  - – Memory copy or clear
  - – Floating point computation

RISC Instruction Set

- Few instructions (typically 32 or 64)
- Each instruction executes in one clock cycle
- Example: MIPS instruction set

Summary Of Instruction Sets

A processor is classified as CISC if the instruction set contains instructions that perform complex computations that can require long times; a processor is classified as RISC if it contains a small number of instructions that can each execute in one clock cycle.

Execution Pipeline

- Hardware optimization technique
- Allows processor to complete instructions faster
- Typically used with RISC instruction set

Typical Instruction Cycle

- Fetch the next instruction
- Examine the opcode to determine how many operands are needed
- Fetch each of the operands (e.g., extract values from registers)
- Perform the operation specified by the opcode
- Store the result in the location specified (e.g., a register)

To Optimize Instruction Cycle

- Build separate hardware block for each step
- Arrange to pass instruction through sequence of hardware blocks

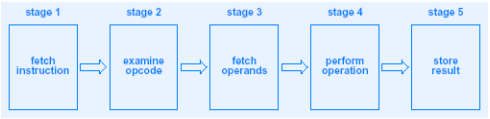


Illustration Of Execution Pipeline (Example pipeline has five stages)

Pipeline Speed

- All stages operate in parallel
- Given stage can start to process a new instruction as soon as current instruction finishes
- Effect: N-stage pipeline can operate on N instructions simultaneously

Illustration Of Instructions In A Pipeline

clock	stage 1	stage 2	stage 3	stage 4	stage 5
1	inst. 1	.	.	.	.
2	inst. 2	inst. 1	.	.	.
3	inst. 3	inst. 2	inst. 1	.	.
4	inst. 4	inst. 3	inst. 2	inst. 1	.
5	inst. 5	inst. 4	inst. 3	inst. 2	inst. 1
6	inst. 6	inst. 5	inst. 4	inst. 3	inst. 2
7	inst. 7	inst. 6	inst. 5	inst. 4	inst. 3
8	inst. 8	inst. 7	inst. 6	inst. 5	inst. 4

Time

RISC Processors And Pipelines

Although a RISC processor cannot perform all steps of the fetch-execute cycle in a single clock cycle, an instruction pipeline with parallel hardware provides approximately the same performance: once the pipeline is full, one instruction completes on every clock cycle.

Using A Pipeline

- Pipeline is *transparent* to programmer
- Disadvantage: programmer who does not understand pipeline can produce inefficient code
- Reason: hardware automatically *stalls* pipeline if items are not available

Example Of Instruction Stalls

- Assume
  - – Need to perform addition and subtraction operations
  - – Operands and results in register *A* through *E*
- – Code is:

  Instruction K: C ← add A B

  Instruction K+1: D ← subtract E C
- Second instruction stalls to wait for operand *C*

A Note About Pipelines

Although hardware that uses an instruction pipeline will not run at full speed unless programs are written to accommodate the pipeline, a programmer can choose to ignore pipelining and assume the hardware will automatically increase speed whenever possible.

No-Op Instructions

- Have no effect on
  - – Registers
  - – Memory
  - – Program counter
  - – Computation
- Documents an instruction stall

Types Of Operations

- One possible categorization
  - – Arithmetic instructions (integer arithmetic)
  - – Logical instructions (also called Boolean)
  - – Data access and transfer instructions
  - – Conditional and unconditional branch instructions
  - – Floating point instructions
  - – Processor control instructions

Program Counter

- Hardware register
- Used during fetch-execute cycle
- Gives address of next instruction to execute
- Also known as *instruction pointer*

Fetch-Execute Algorithm Details

Assign the program counter an initial program address.

Repeat forever {

Fetch:

Access the next step of the program from the location given by the program counter.

Set an internal address register, A, to the address beyond the instruction that was just fetched.

Execute:

Perform the step of the program.

Copy the contents of address register A to the program counter.

}

Example Instruction Set

- Known as *MIPS* instruction set
- Early RISC design
- Minimalistic

MIPS Instruction Set (Part 1)

Instruction	Meaning
Arithmetic	
add	integer addition
subtract	integer subtraction
add immediate	integer addition (register + constant)
add unsigned	unsigned integer addition
subtract unsigned	unsigned integer subtraction
add immediate unsigned	unsigned addition with a constant
move from coprocessor	access coprocessor register
multiply	integer multiplication
multiply unsigned	unsigned integer multiplication
divide	integer division
divide unsigned	unsigned integer division
move from Hi	access high-order register
move from Lo	access low-order register
Logical (Boolean)	
and	logical *and* (two registers)
or	logical *or* (two registers)
and immediate	*and* of register and constant
or immediate	*or* of register and constant
shift left logical	Shift register left N bits
shift right logical	Shift register right N bits

MIPS Instruction Set (Part 2)

Instruction	Meaning
Data Transfer	
load word	load register from memory
store word	store register into memory
load upper immediate	place constant in upper sixteen bits of register
move from coproc. register	obtain a value from a coprocessor
Conditional Branch	
branch equal	branch if two registers equal
branch not equal	branch if two registers unequal
set on less than	compare two registers
set less than immediate	compare register and constant
set less than unsigned	compare unsigned registers
set less than immediate	compare unsigned register and constant
Unconditional Branch	
jump	go to target address
jump register	go to address in register
jump and link	procedure call

MIPS Floating Point Instructions

Instruction	Meaning
Arithmetic	
FP add	floating point addition
FP subtract	floating point subtraction
FP multiply	floating point multiplication
FP divide	floating point division
FP add double	double-precision addition
FP subtract double	double-precision subtraction
FP multiply double	double-precision multiplication
FP divide double	double-precision division
Data Transfer	
load word coprocessor	load value into FP register
store word coprocessor	store FP register to memory
Conditional Branch	
branch FP true	branch if FP condition is true
branch FP false	branch if FP condition is false
FP compare single	compare two FP registers
FP compare double	compare two double precision values

Aesthetic Aspects Of Instruction Set

- Elegance
  - – Balanced
  - – No frivolous or useless instructions
- Orthogonality
  - – No unnecessary duplication
  - – No overlap among instructions

Principle Of Orthogonality

The principle of orthogonality specifies that each instruction should perform a unique task without duplicating or overlapping the functionality of other instructions.

Addresses in an Instruction (I)

- In a typical arithmetic or logical instruction, 3 addresses are required
  - – 2 operands and 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

Addresses in an Instruction (II)

- 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
- 1 address instructions
  - –Two addresses are implied in the instruction
  - Traditional accumulator-based operations

    Example: Acc = Acc + X

Addresses in an Instruction (III)

- 0 address instructions
  - All addresses are implied, as in register-based operations

    Example: 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

Addresses in an Instruction (IV)

- Trade off:
  - Fewer addresses in the instruction results in
    - – More primitive instructions
    - – Less complex CPU
    - – Instructions with shorter length
    - – More total instructions in a program
    - – Longer, more complex programs
    - – Longer execution time

Addresses in an Instruction (V)

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

3 address

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

2 address

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

1 address

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


Addressing Mode

- Once we have determined the number of addresses contained in an instruction, the manner in which each address field specifies memory location must be determined
- Want the ability to reference a large range of address locations
- Tradeoff between
  - Addressing range and flexibility
  - Complexity of the address calculation

Addressing Mode: Immediate Mode

- The operand is contained within the instruction itself
- Data is a constant at run time
- No additional memory references are required after the fetch of the instruction itself
- Fast, but size of the operand (thus its range of values) is limited

  e.g. ADD 5

  Add 5 to contents of accumulator

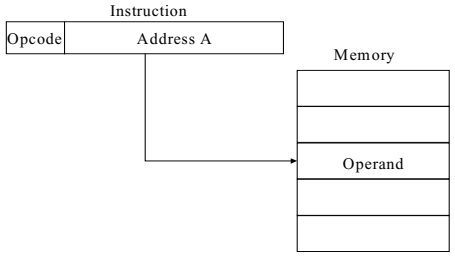
  5 is operand

Instruction Opcode Operand

Addressing Mode: Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

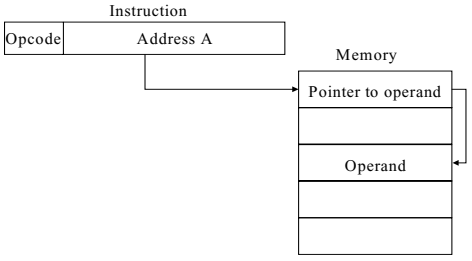
Addressing Mode: Direct Addressing



Addressing Mode: Indirect Addressing

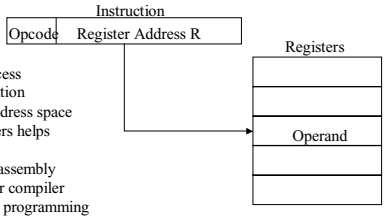
- The address field in the instruction specifies a memory location which contains the address of the data
  - Two memory accesses are required
  - The first to fetch the effective address
  - The second to fetch the operand itself
- Range of effective addresses is equal to 2^n ,where n is the width of the memory data word
- Number of locations that can be used to hold the effective address is constrained to 2^k , where k is the width of the instruction's address field

Addressing Mode: Indirect Addressing



Addressing Mode: Register Addressing

- Register addressing: like direct, but address field specifies a register location



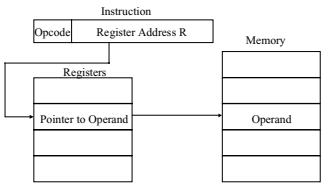
No memory access
Very fast execution
Very limited address space
Multiple registers helps performance
Requires good assembly programming or compiler writing, N.B. C programming

Addressing Mode: Register Addressing

- Register indirect: like indirect, but address field specifies a register that contains the effective address

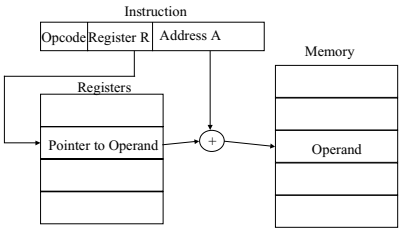
Large address space (2^n)

One fewer memory access than indirect addressing



Addressing Mode: Displacement Addressing

EA = A + (R); Address field hold two values
A = base value; R = register that holds displacement or vice versa



Motorola 68000

Programmer's Model

This is the greatly simplified view of how a 68000 processor works which is all that a programmer really needs to know in order to write 68000 assembly language programs.

8 32-bit **data registers**, named D0, D1, .. , D7
7 32-bit **address registers**, named A0, A1, .. ,A6
a special 32-bit address register A7, used as a stack pointer
a 32-bit **program counter (PC)** register
a 16-bit **status register**

Data can be manipulated in chunks of:
1 bit; 8 bits(byte); 16 bits (word); 32 bits (longword)

Motorola 68000

MOVE

This is used for *copying* data from one register to another, or between registers and main memory.

eg

```
MOVE.B    D1,D2
MOVE.W    (A1),D3
MOVE.L    #10,D0
MOVE.B    D0,10000
MOVE.L    $1000,D5
```

.B -> Byte; .W -> Word; L -> Longword

Motorola 68000

ADD

This adds integers stored in registers or in memory

eg

```
ADD.B     D1,D2
ADD.B     #10,D2
```

At least one of the operands must be a register. The result is left in the destination operand.

Motorola 68000

An example program

The following short program adds two byte-sized numbers.Numbers to be added are initially stored in memory. Numbers are at addresses $400420 and $400422. Answer will be stored at memory address $400424

```
ORG     $400400          Set start address of the program

START   MOVE.B    $400420, D0     Move first number to D0
        ADD.B     $400422, D0     Add second number to first
        MOVE.B    D0, $400424     Store answer in memory
STOP    BRA       STOP            "Stop" the program
        END
```

BRA: The branch instruction, used to jump to a named instruction somewhere in the program

Intel Pentium Processor - Memory Organization

The memory on the bus of a Pentium processor is called *physical memory*.

It is organized as a sequence of 8-bit bytes.

Each byte is assigned a unique address, called a *physical address*, which ranges from zero to a maximum of $2^{32}-1$ (4 gigabytes).

Memory can appear as a single, "flat" address space like physical memory.

Or, it can appear as one or more independent memory spaces, called *segments*.

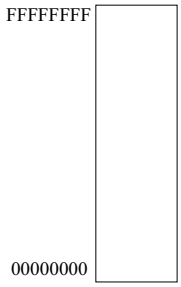
Segments can be assigned specifically for holding a program's code (instructions), data, or stack

Intel Pentium Processor - Memory Organization

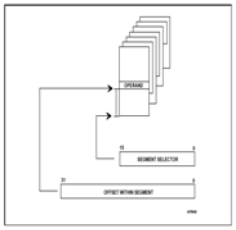
Un-segmented or "Flat" Model

The simplest memory model is the flat model.

In a flat model, segments can cover the entire range of physical addresses, or they can cover only those addresses which are mapped to physical memory.



Intel Pentium Processor - Memory Organization



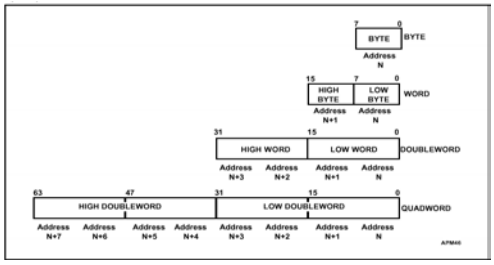
The logical address space

16,383 segments, to 4 gigabytes each
Total 2^{46} bytes (64 terabytes).

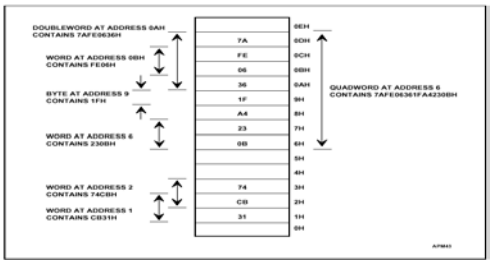
The processor maps this 64 terabyte logical address space onto the physical address space by the address translation mechanism.

A pointer into a segmented address space consists of two parts
1. A *segment selector*, which is a 16-bit field which identifies a segment.
2. An *offset*, which is a 32-bit byte address within a segment.

Intel Pentium Processor - Data Types



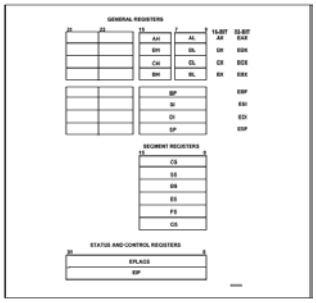
Intel Pentium Processor - Byte Ordering



Intel Pentium Processor - Registers

- The processor contains sixteen registers which can be used by an application programmer. As

1. General registers. These eight 32-bit registers are free for use by the programmer.
2. Segment registers. These registers hold segment selectors associated with different forms of memory access.
3. Status and control registers. These registers report and allow modification of the state of the processor.



Intel Pentium Processor - Registers

General Registers

Eight 32-bit registers EAX, EBX, ECX, EDX, EBP, ESP, ESI, and EDI.

Operands for logical and arithmetic operations.

Operands for address calculations

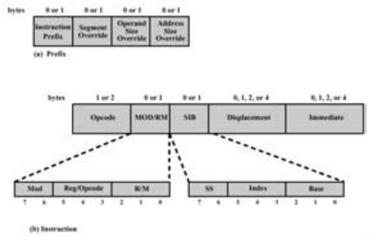
Can be access as 8, 16, or 32 bit chunks.

All are available for address calculations and for the results of most arithmetic and logical operations

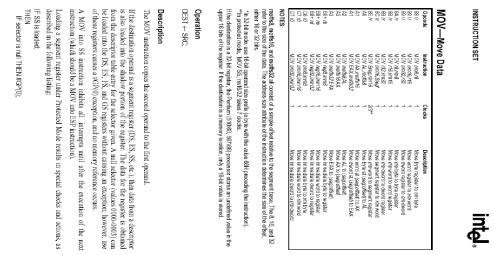
A few instructions assign specific registers to hold operands so that the instruction set can be encoded more compactly.

Intel Pentium Processor - Instruction format

The information encoded in an instruction includes a specification of the operation to be performed the type of the operands to be manipulated, and the location of these operands.



Intel Pentium Processor - Instruction format



Intel Pentium Processor - Instruction Examples

ADD

ADD AL, imm8	Add immediate byte to AL
ADD AX, imm16	Add immediate word to AX
ADD EAX, imm32	Add immediate dword to EAX

Operation
DEST = DEST + SRC;

Description
The ADD instruction performs an integer addition of the two operands (DEST and SRC). The result of the addition is assigned to the first operand (DEST), and the flags are set accordingly.

Intel Pentium Processor - Instruction Examples

ADD r/m8,imm8	Add immediate byte to r/m byte
ADD r/m16,imm16	Add immediate word to r/m word
ADD r/m32,imm32	Add immediate dword to r/m dword
ADD r/m8,r8	Add byte register to r/m byte
ADD r/m16,r16	Add word register to r/m word
ADD r/m32,r32	Add dword register to r/m dword

r/m8: a one-byte operand that is either the contents of a byte register (AL, BL, CL, DL, AH, BH, CH, DH), or a byte from memory.
r/m16: a word register or memory operand used for instructions whose operand-size attribute is 16 bits. The word registers are: AX, BX, CX, DX, SP, BP, SI, DI. The contents of memory are found at the address provided by the effective address computation.
r/m32: a doubleword register or memory operand used for instructions whose operand-size attribute is 32 bits. The doubleword registers are: EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI. The contents of memory are found at the address provided by the effective address computation.