Instruction Set Architecture

Outline

- **An Instruction**
- **Instruction Set Architecture**
- **Classification of ISAs**
- **Classification of Operations**
- **Operands**
- **Instruction Format**
- **Addressing Modes**
- **Evolution of the Instruction Set**
- **RISC Versus CISC**
- **MIPS Instruction Set**

An Instruction

- An instruction can be considered as a **word** in processor's language
- What information an Instruction should covey to the CPU?

opcode	Addr of OP1	Addr of OP2	Dest Addr	Addr of next intr

- The instruction is too long if everything specified explicitly
  - More space in memory
  - Longer execution time
- How can you reduced the size of an instruction?
  - Specifying information implicitly
  - How?
- Using PC, ACC, GPRs, SP

ISA "Instruction Set Architecture is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct program for that machine."

❑ The ISA defines:

- **Operations that the processor can execute**
- **Data Transfer mechanisms + how to access data**
- **Control Mechanisms (branch, jump, etc)**
- **"Contract" between programmer/compiler and HW**

❑ISA is important:

- **Not only from the programmer's perspective.**
- **From processor design and implementation perspectives as well.**

Programmer visible part of a processor:

- Registers (where are data located?)
- Addressing Modes (how is data accessed?)
- Instruction Format (how are instructions specified?)
- Exceptional Conditions (what happens if something goes wrong?)
- Instruction Set (what operations can be performed?)

ISA Design Choices

- **Types of operations supported**
  – e.g. arithmetic/logical, data transfer, control transfer, system, floating-point, decimal (BCD), string

- **Types of operands supported**
  – e.g. byte, character, digit, halfword, word (32-bits), doubleword, floating-point number

- **Types of operand storage allowed**
  – e.g. stack, accumulator, registers, memory

- **Implicit vs. explicit operands in instructions and number of each**
- **Orthogonality of operands, operand location, and addressing modes**

Classification of ISAs

Determined by the means used for storing data in CPU:

The major choices are:

- A stack, an accumulator, or a set of registers

Stack architecture:

- Operands are implicitly on top of the stack

Accumulator architecture:

- One operand is in the accumulator (register) and the others are elsewhere
- Essentially this is a 1 register machine
- Found in older machines…

General purpose registers:

- Operands are in registers or specific memory locations.

Evolution of ISAs

- **Accumulator Architectures (EDSAC, IBM-701)**
- **Special-purpose Register Architectures**
- **General purpose registers architectures**
  - **Register-memory (IBM 360, DEC PDP-11, Intel 80386**
  - **Register-register (load-store) (CDC6600, MIPS, DEC alpha)**

Single Accumulator

**(Manchester Mark I,
IBM 700 series 1953)**

Stack

(Burroughs, HP-3000 1960-70)

General Purpose Register Machines

Complex Instruction Sets

(Vax, Intel 386 1977-85)

RISC

(MIPS, IBM RS6000, . . .1987)

Classification of ISAs

Types of Architecture	Source Operands	Destination
Stack	Top two elements in stack	Top of stack
Accumulator	Accumulator (1) Memory (other)	Accumulator
Register set	Register or Memory	Register or Memory

Comparison of Architectures

Consider the operation: C = A + B

Stack	Accumulator	Register-Memory	Register-Register
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3

Classification of Operations

Data Transfer – Load, store, mov

Data Manipulation

- Arithmetic – Add, subtract, multiply, divide
- Signed, unsigned, integer, floating-point
- Logical - Conjunction, disjunction, shift left, shift right

Status manipulation

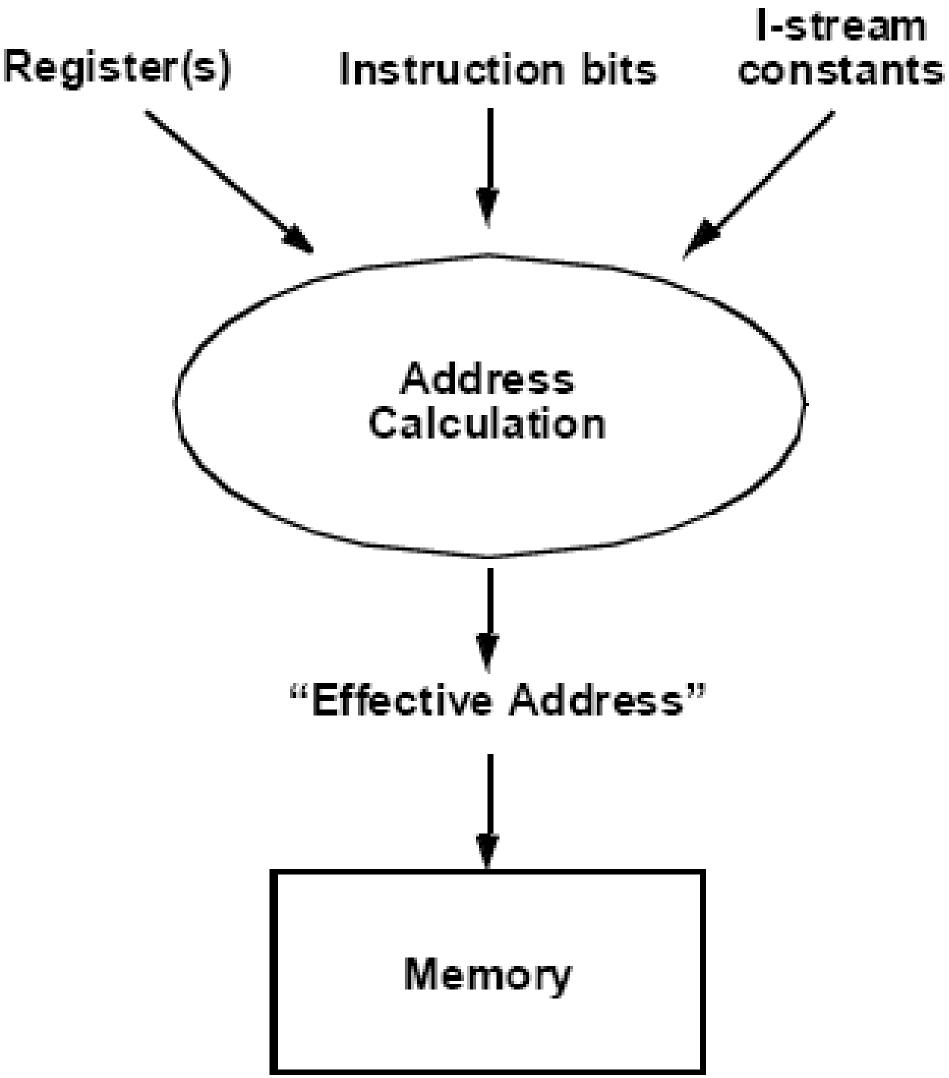
Control Transfer

- Conditional Branch
- Branch on equal, not equal
- Set on less than
- Unconditional Jump

Addressing Modes

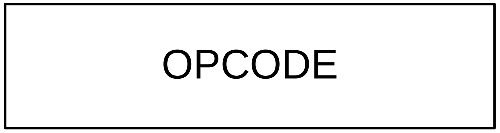
Addressing modes describe how an instruction can find the location of its operands

- **Effective address = actual memory address specified by an addressing mode**

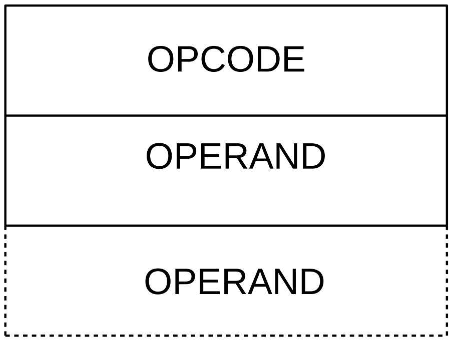


Addressing Modes

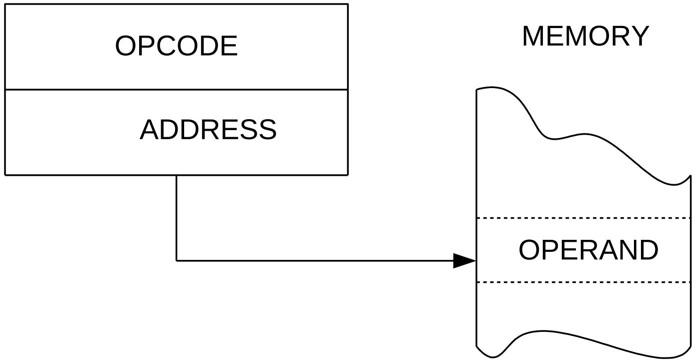
INHERENT (0-address)



IMMEDIATE

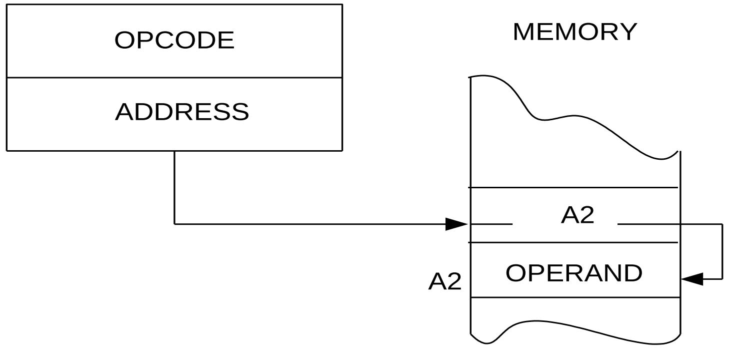


ABSOLUTE (DIRECT)

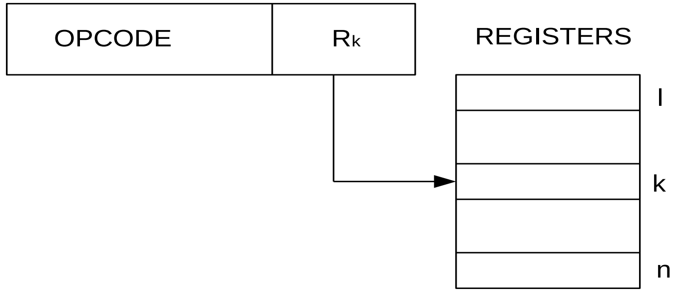


Addressing Modes

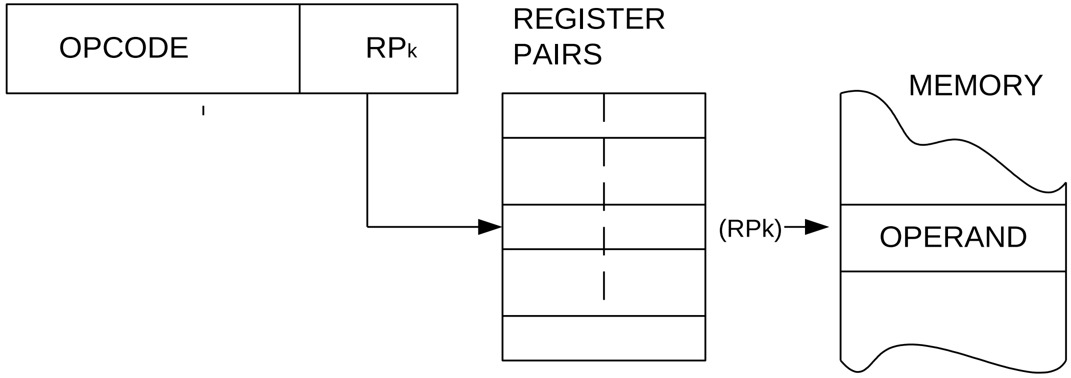
INDIRECT



REGISTER

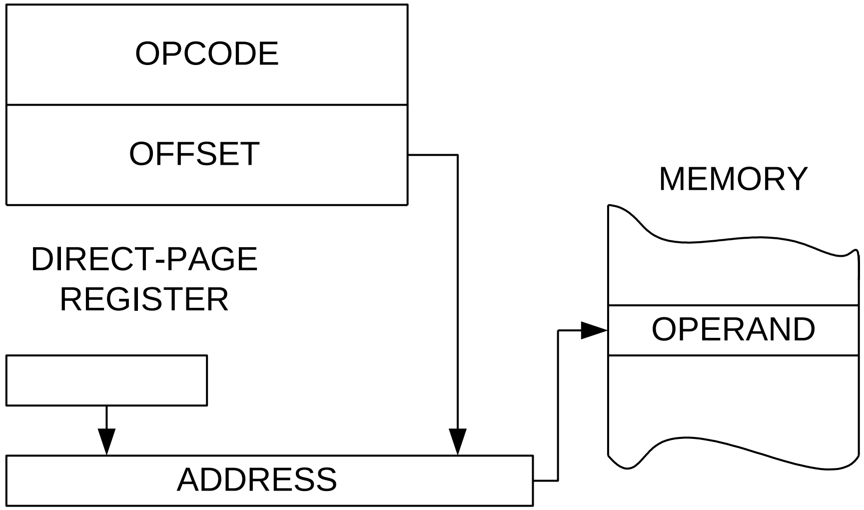


REGISTER INDIRECT

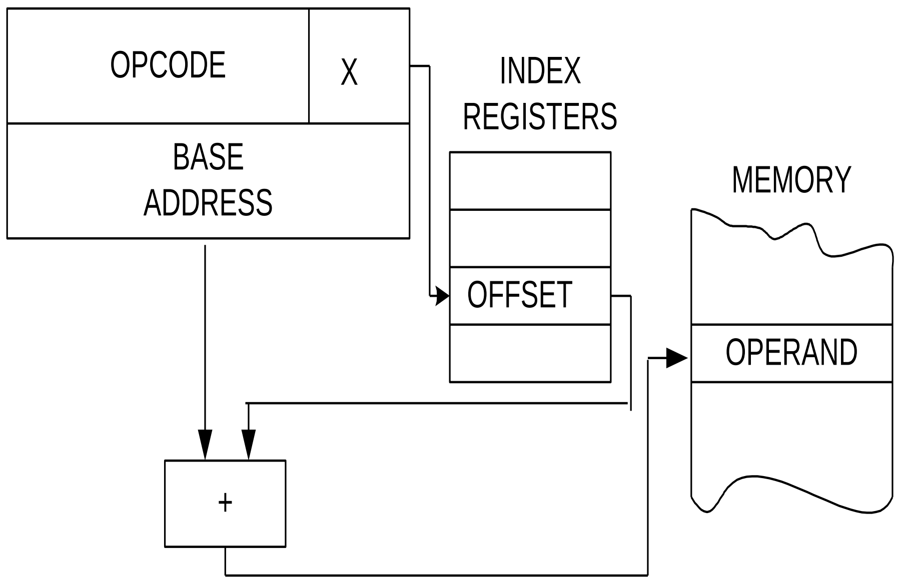


Addressing Modes

PAGED



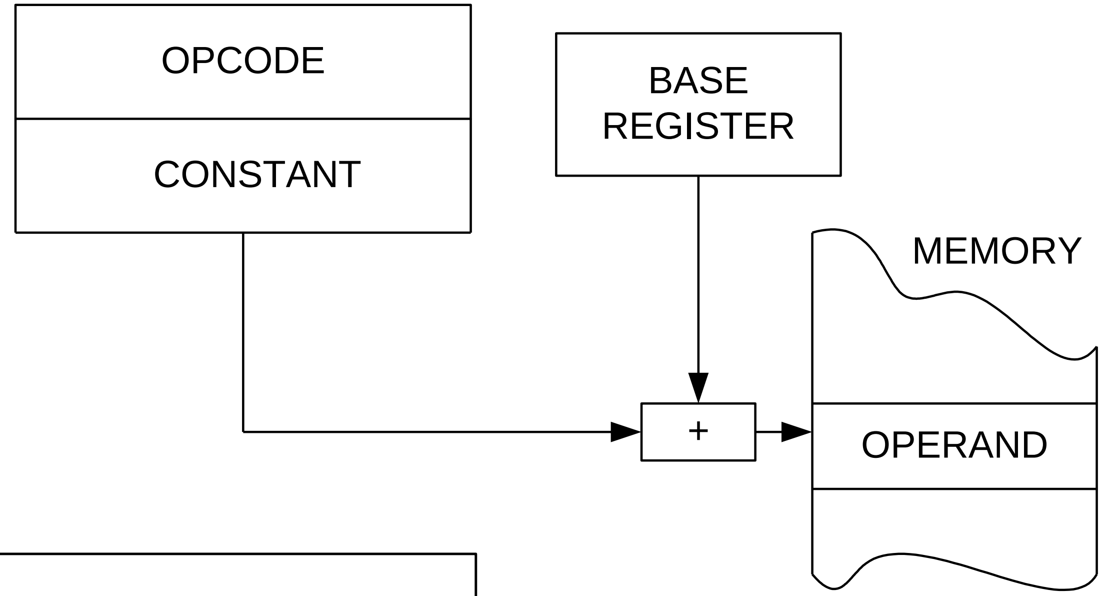
INDEXED



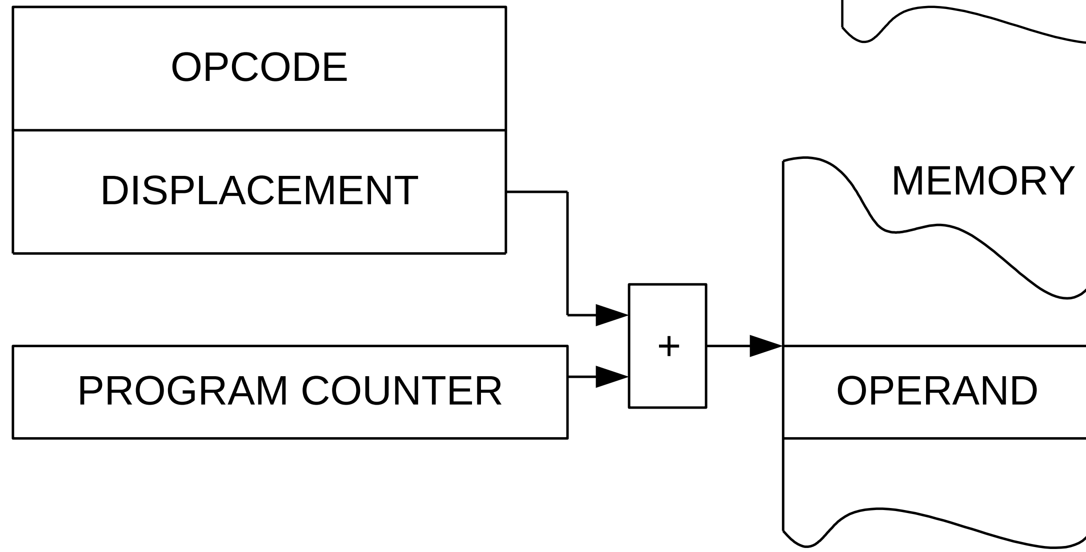
Addressing Modes

BASED

BASED-INDEXED

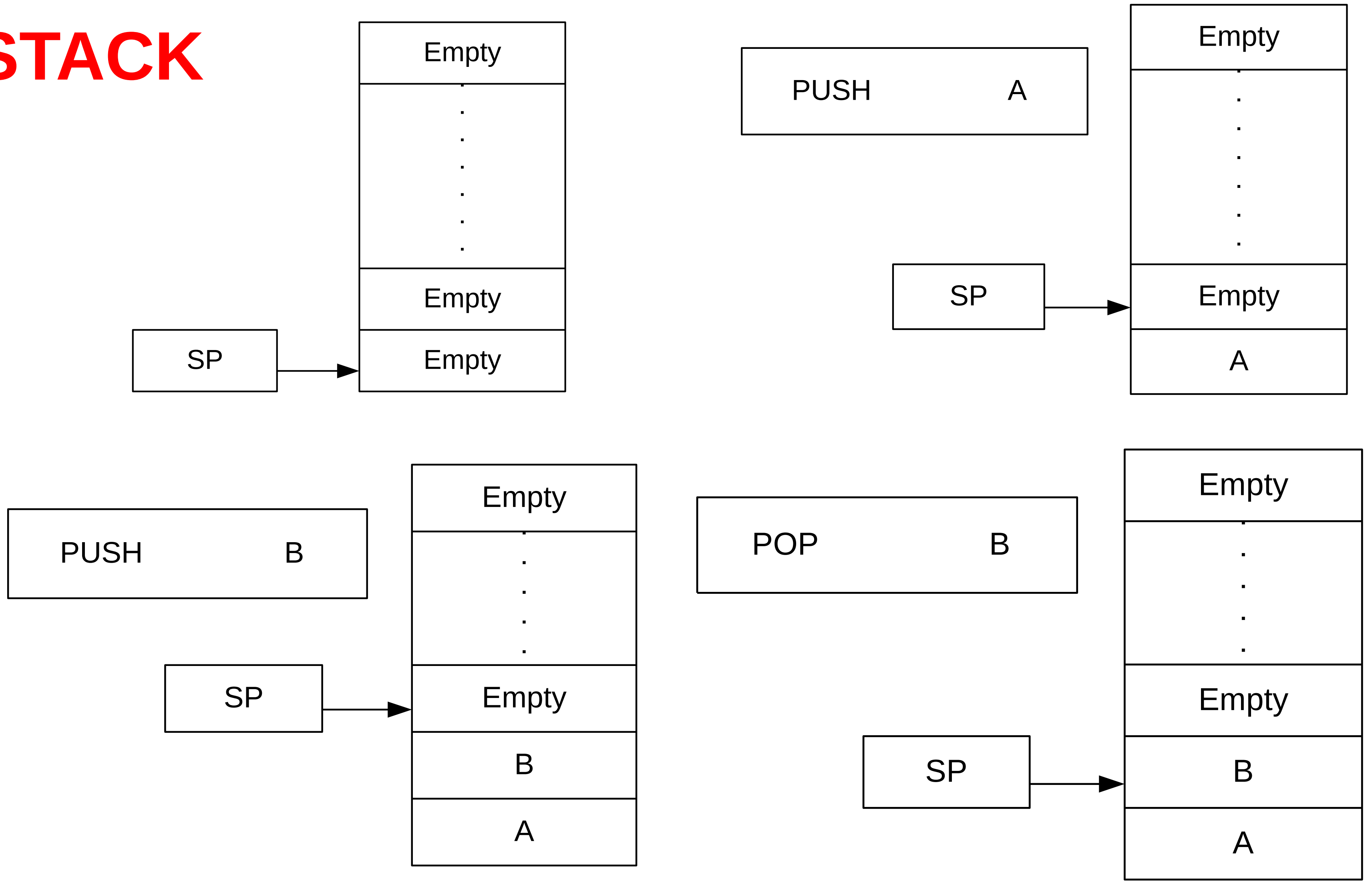


RELATIVE



Addressing Modes

STACK



Case Study

MIPS Instruction Set architecture

Operands in MIPS

❑ Registers

– MIPS has 32 architected 32-bit registers
– Effective use of registers is key to program performance

❑ Data Transfer

– 32 words in registers, millions of words in main memory
– Instruction accesses memory via memory address
– Address indexes memory, a large single-dimensional array

❑ Alignment

– Most architectures address individual bytes
– Addresses of sequential words differ by 4
– Words must always start at addresses that are multiples of 4

MSB	*byte-3*	*byte-2*	*byte-1*	*byte-0*	*LSB*
	byte-7	*byte-6*	*byte-5*	*byte-4*	

Register Usage Conventions

❑ Registers

– MIPS has 32 architected 32-bit registers
– Effective use of registers is key to program performance

- r0: always 0
- r1: reserved for the assembler
- r2, r3: function return values
- r4~r7: function call arguments
- r8~r15: “caller-saved” temporaries
- r16~r23 “callee-saved” temporaries
- r24~r25 “caller-saved” temporaries
- r26, r27: reserved for the operating system
- r28: global pointer
- r29: stack pointer
- r30: callee-saved temporaries
- r31: return address

Data Types in MIPS

Data and instructions are both represented in bits

- **32-bit architectures employ 32-bit instructions**
- **Combination of fields specifying operations/operands**

➢ **MIPS operates on:**

- **32-bit (unsigned or 2's complement) integers,**
- **32-bit (single precision floating point) real numbers,**
- **64-bit (double precision floating point) real numbers;**
- **32-bit words, bytes and half words can be loaded into GPRs**
- **After loading into GPRs, bytes and half words are either zero or sign bit expanded to fill the 32 bits;**
- **Only 32-bit units can be loaded into FPRs and 32-bit real numbers are stored in even numbered FPRs.**
- **64-bit real numbers are stored in two consecutive FPRs, starting with even-numbered register.**
- **Floating-point numbers IEEE standard 754 float: 8-bit exponent, 23-bit significand double:11-bit exponent, 52-bit significand**

MIPS Memory Organization

- **• MIPS supports byte addressability:**

▪ it implies that a byte is the smallest unit with its address;

▪ 32-bit word has to start at byte address that is multiple of 4;

▪ Thus, 32-bit word at address 4n includes four bytes with addresses: 4n, 4n+1, 4n+2, and 4n+3.

▪16-bit half word has to start at byte address that is multiple of 2; Thus, 16-bit word at address 2n includes two bytes with addresses: 2n and 2n+1.

▪ it implies that an address is given as 32-bit unsigned

MIPS Instruction Formats

R-Type Instruction Fields

– op: basic operation of instruction, called the *opcode*
– rs, rt: first, second register source operand
– rd: register destination operand
– shamt: shift amount for shift instructions
– funct: specifies a variant of the operation, called *function code*

op	rs	rt	rt	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

MIPS Instruction Formats

- **I-Type Instruction Fields**

– Opcode specifies instruction format

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

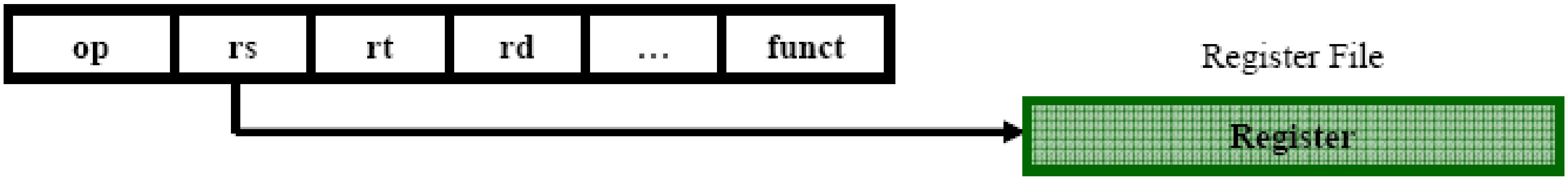
- **J-Type Instruction Fields**

op	address
6 bits	26 bits

MIPS Addressing Modes

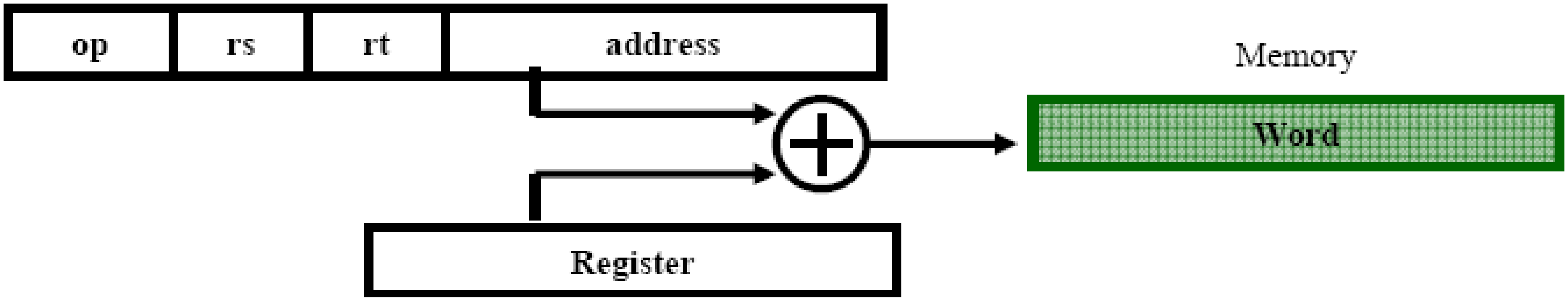
- **Register Addressing**

– Operand is a register (e.g., add $s2, $s0, $s1)



- **Base or Displacement Addressing**

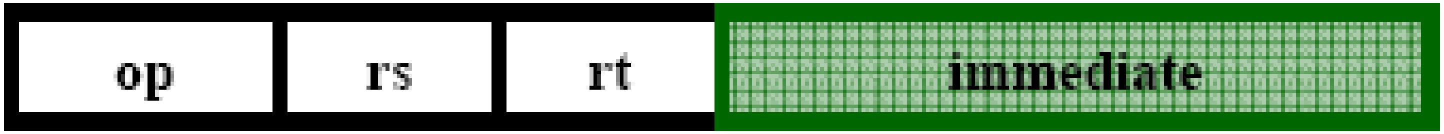
– Operand is at memory location whose address is sum of register and constant (e.g., lw $s1, 8($s0))



MIPS Addressing Modes

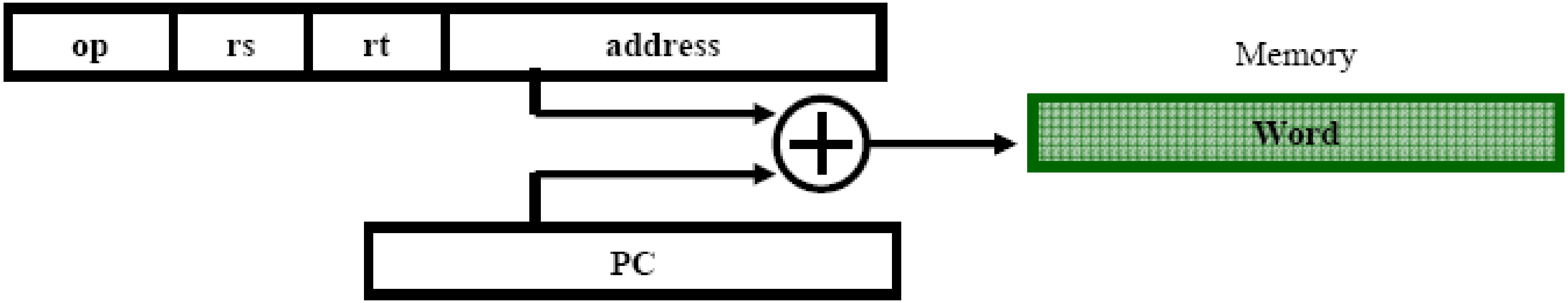
- **Immediate Addressing**

– Operand is constant within instruction (e.g., addi $s1, $s0, 4)



- **PC-Relative Addressing**

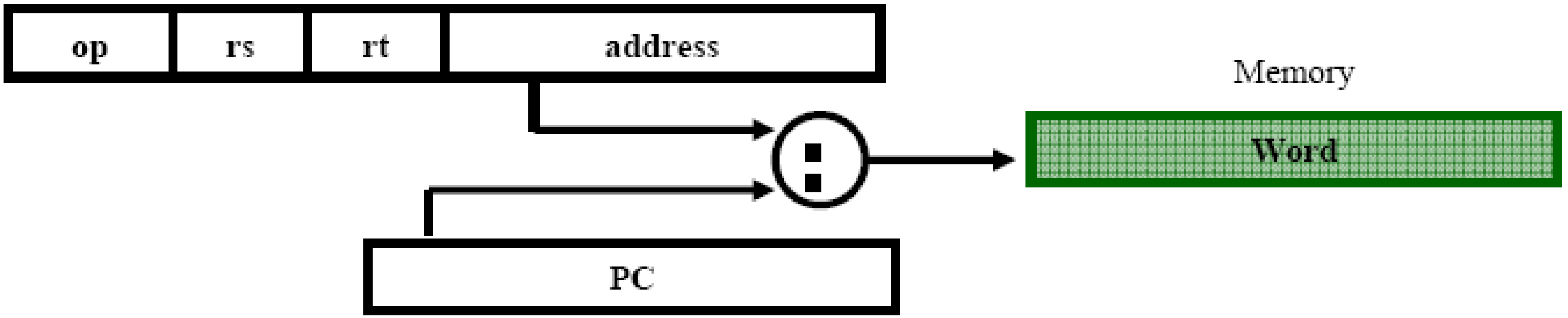
– Address is sum of PC and constant within instruction (e.g., beq $s0, $s1, 16)



MIPS Addressing Modes

- **Pseudo-direct Addressing**

– Address is 26 bits of constant within instruction concatenated with upper 6 bits of PC (e.g., j 1000)



Survey of MIPS Instruction Set

- **Arithmetic**

- **Addition**

– add $s1, $s2, $s3 # $s1 = $2 + $s3, overflow detected
– addu $s1, $s2, $s3 # $s1 = $s2 x $s3, overflow undetected

- **Subtraction**

– sub $s1, $s2, $s3 # $s1 = $s2 - $s3, overflow detected
– subu $s1, $s2, $s3 # $s1 = $s2 - $s3, overflow undetected

- **Immediate/Constants**

– addi $s1, $s2, 100 # $s1 = $s2 + 100, overflow detected
– addiu $s1, $s2, 100 # $s1 = $s2 + 100, overflow undetected

Survey of MIPS Instruction Set

- Arithmetic

- **Multiply**

– mult $s2, $s3 # Hi, Lo = $s2 x $s3, 64-bit signed product
– multu $s2, $s3 # Hi, Lo = $s2 x $s3, 64-bit unsigned product

- **Divide**

– div $s2, $s3 # Lo = $s2/$s3, Hi = $s2 mod $s3
– divu $s2, $s3 # Lo = $s2/$s3, Hi = $s2 mod $s3, unsigned

- **Ancillary**

– mfhi $s1 # $s1 = Hi, get copy of Hi
– mflo $s1 # $s1 = Lo, get copy of low

Survey of MIPS Instruction Set

- **Logical**

  - **Boolean Operations**
  - – and $s1, $s2, $s3 # $s1 = $s2 & $s3, bit-wise AND
  - – or $s1, $s2, $s3 # $s1 = $s2 | $s3, bit-wise OR

  - **Immediate/Constants**
  - – andi $s1, $s2, 100 # $s1 = $s2 & 100, bit-wise AND
  - – ori $s1, $s2, 100 # $s1 = $s2 | 100, bit-wise OR

  - **Shifting**
  - – sll $s1, $s2, 10 # $s1 = $s2 << 10, shift left
  - – srl $s1, $s2, 10 # $s1 = $s2 >> 100, shift right

Survey of MIPS Instruction Set

- **Data Transfer**

- **Load Operations**
  - lw $s1, 100($s2) # $s1 = Mem($s2+100), load word
  - lbu $s1, 100($s2) # $s1 = Mem($s2+100), load byte
  - lui $s1, 100 # $s1 = 100 * 2^16, load upper imm.

- **Store Operations**
  - sw $s1, 100($s2) # Mem[$s2+100] = $s1, store word
  - sb $s1, 100($s2) # Mem[$s2+100] = $s1, store byte

Survey of MIPS Instruction Set

- Control Transfer

- Jump Operations
– j 2500 # go to 10000
– jal 2500 # $ra = PC+4, go to 10000, for procedure call
– jr $ra # go to $ra, for procedure return

- Branch Operations
– beq $s1, $s2, 25 # if($s1==$s2), go to PC+4+100, PC-relative
– bne $s1, $s2, 25 # if($s1!=$s2), go to PC+4+100, PC-relative

- Comparison Operations
– slt $s1, $s2, $s3 # if($s2<$s3) $s1=1, else $s1=0, 2's comp.
– slti $s1, $s2, 100 # if($s2<100), $s1=1, else $s1=0, 2's comp.
– sltu $s1, $s2, $s3 # if($s2<$s3), $s1=1, else $s1=0, unsigned
– sltiu $s1, $s2, 100 # if($s2<100) $s1=1, else $s1=0, unsigned

Survey of MIPS Instruction Set

- **Floating Point Operations**

- **Arithmetic**

– {add.s, sub.s, mul.s, div.s} $f2, $f4, $f6 # single-precision
– {add.d, sub.d, mul.d, div.d} $f2, $f4, $f6 # double-precision
– Floating-point registers are used in even-odd pairs, using even number register as its name

- **Data Transfer**

– {lwc1, swc1} $f1, 100($s2)
– Transfer data to/from floating-point register file

- **Conditional Branch**

– {c.lt.s, c.lt.d} $f2, $f4 # if ($f2 < $f4), cond=1, else cond=0
– {bclt, bclf} 25 # if (cond==1/0), go to PC+4+100

Thanks!

Q & A