Instruction Set Architecture Design

Mark McDermott

Fall 2009

Agenda

- What are ISAs
- Evolution of Instruction Sets
- Types of ISAs According To Operand Addressing Fields
- Addressing modes
- Factors affecting the design of ISAs

Instruction Set Architecture (ISA)

"... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."

Amdahl, Blaauw, and Brooks, 1964.

The instruction set architecture is concerned with:

- Organization of programmable storage (memory & registers): Includes the amount of addressable memory and number of available registers.
- Data Types & Data Structures: Encodings & representations.
- Instruction Set: What operations are specified.
- Instruction formats and encoding.
- Modes of addressing and accessing data items and instructions
- Exception conditions.

Computer Instruction Sets

- **Regardless of computer type, CPU structure, or hardware organization, every machine instruction must specify the following:**
  - **Opcode: Which operation to perform. Example: add, load, and branch.**
  - **Where to find the operand or operands, if any: Operands may be contained in CPU registers, main memory, or I/O ports.**
  - **Where to put the result, if there is a result: May be explicitly mentioned or implicit in the opcode.**
  - **Where to find the next instruction: Without any explicit branches, the instruction to execute is the next instruction in the sequence or a specified address in case of jump or branch instructions.**

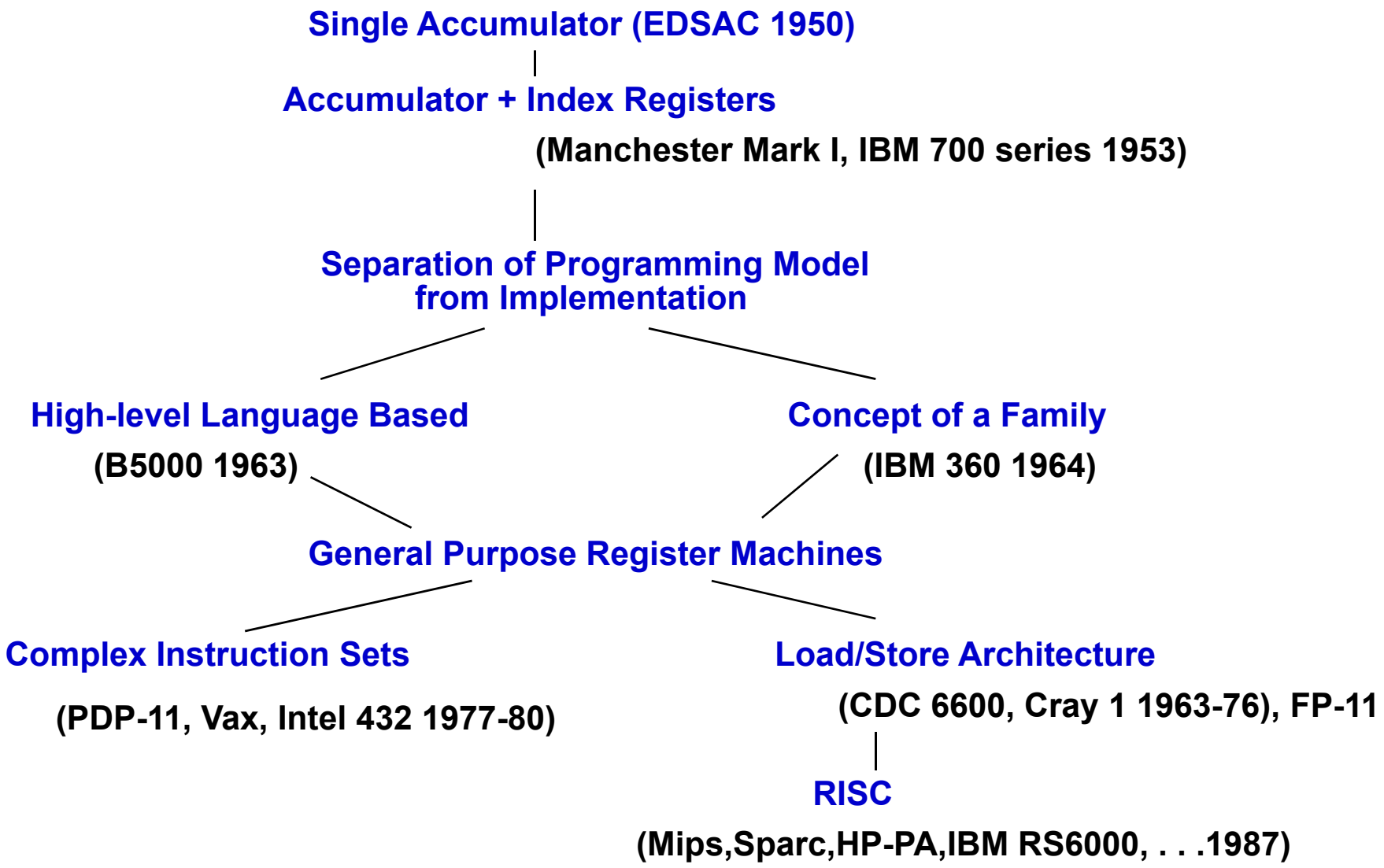
Main General Types of Instructions

- **Data Movement Instructions, possible variations:**
  - Memory-to-memory.
  - Memory-to-CPU register.
  - CPU-to-memory.
  - Constant-to-CPU register.
  - CPU-to-output.
  - etc.
- **Arithmetic Logic Unit (ALU) Instructions:**
  - Logic instructions
  - Integer Arithmetic Instructions
  - Floating Point Arithmetic Instructions
- **Branch (Control) Instructions:**
  - Unconditional jumps.
  - Conditional branches.

Operation Types in The Instruction Set

Operator Type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, or
Data transfer	Loads-stores (move on machines with memory addressing)
Control	Branch, jump, procedure call, and return, traps.
System	Operating system call/return, virtual memory management instructions ...
Floating Point	Floating point operations: add, multiply, divide
Decimal	Decimal add, decimal multiply, decimal to character conversion
String	String move, string compare, string search
Media	The same operation performed on multiple data (e.g Intel MMX, SSE)

Evolution of Instruction Sets



ISA Examples

Machine	Number of General Purpose Registers	Architecture	year
EDSAC	1	accumulator	1949
IBM 701	1	accumulator	1953
CDC 6600	8	load-store	1963
IBM 360	16	register-memory	1964
DEC PDP-8	1	accumulator	1965
DEC PDP-11	8	register-memory	1970
Intel 8008	1	accumulator	1972
Motorola 6800	1	accumulator	1974
DEC VAX	16	register-memory memory-memory	1977
Intel 8086	1	extended accumulator	1978
Motorola 68000	16	register-memory	1980
Intel 80386	8	register-memory	1985
MIPS	32	load-store	1985
HP PA-RISC	32	load-store	1986
SPARC	32	load-store	1987
PowerPC	32	load-store	1992
DEC Alpha	32	load-store	1992
HP/Intel IA-64	128	load-store	2001
AMD64 (EMT64)	16	register-memory	2003

Requisite RISC vs. CISC discussion

- **Reduced instruction set computer (RISC):**
  - load/store;
    - operands in memory must be first loaded into register before any operation
  - 32-bit fixed format instruction (3 formats)
  - 32 32-bit General Purpose Registers
    - (R0 contains zero, DP take pair of Registers)
  - 3-address, reg-reg arithmetic instruction
  - Single address mode for load/store: base + displacement
    - No indirection
  - Simple branch conditions
  - Examples:
    - ARM, MIPS, Sun SPARC, PowerPC
- **Complex instruction set computer (CISC):**
  - many addressing modes;
    - can directly operate on operands in memory
  - many operations.
  - variable instruction length
  - Examples:
    - Intel x86, 68000, VAX microprocessors and compatibles

Instruction Set Architecture: What Must be Specified?

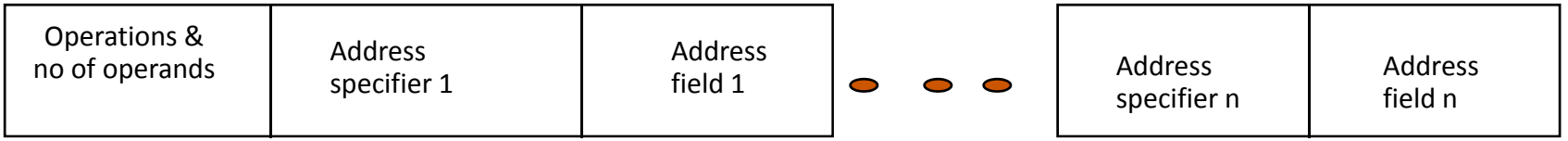
- Instruction Format or Encoding
  - How is it decoded?
- Addressing of operands and result
  - Immediate addressing
  - Register addressing
  - Direct addressing
  - Register indirect addressing
  - Indexed addressing
  - Based-indexed addressing
  - PC Relative
  - Stack addressing
- Data type and Size
  - Fixed, Float, Vector
  - 8, 16, 32, 64 bit
  - Little Endian vs. Big Endian
- Operations
  - ADD, MUL, DIV, SHIFT, OR, AND
- Successor instruction
  - Jumps, conditions, branches
  - Fetch-decode-execute is implicit! – Von Neumann

Instruction Format and Encoding

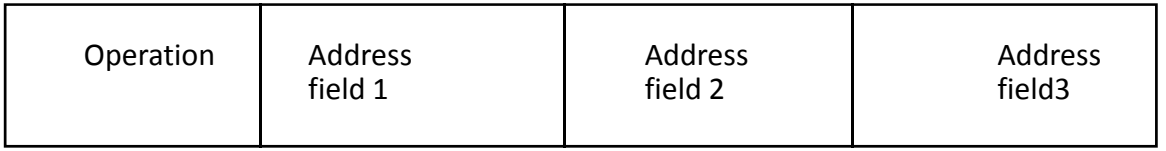
- **Considerations affecting instruction set encoding:**
  - The number of registers and addressing modes supported by ISA.
  - The impact of the size of the register and addressing mode fields on the average instruction size and on the average program.
  - To encode instructions into lengths that will be easy to handle in the implementation. On a minimum to be a multiple of bytes (8 bits).
- **Instruction Encoding Classification:**
  - Fixed length encoding: Faster and easiest to implement in hardware.
  - Variable length encoding: Produces smaller instructions.
  - Hybrid encoding.

Three Examples of Instruction Set Encoding

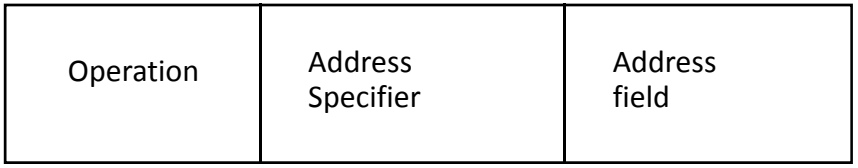
Variable Length Encoding: VAX (1-53 bytes)

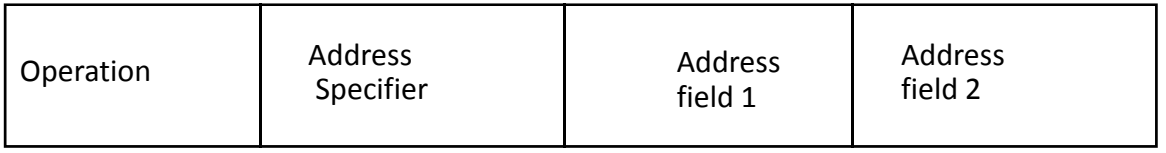


Fixed Length Encoding: MIPS, PowerPC, SPARC (all instructions are 4 bytes each)



Hybrid Encoding: IBM 360/370, Intel 80x86





ARM Instruction Set Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction Type
Condition				0	0	I	OPCODE				S	Rn				Rs				OPERAND-2												Data processing
Condition				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply
Condition				0	0	0	0	1	U	A	S	Rd HIGH				Rd LOW				Rs				1	0	0	1	Rm				Long Multiply
Condition				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				Swap
Condition				0	1	I	P	U	B	W	L	Rn				Rd				OFFSET												Load/Store - Byte/Word
Condition				1	0	0	P	U	B	W	L	Rn				REGISTER LIST																Load/Store Multiple
Condition				0	0	0	P	U	1	W	L	Rn				Rd				OFFSET 1				1	S	H	1	OFFSET 2				Halfword Transfer Imm Off
Condition				0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				Halfword Transfer Reg Off
Condition				1	0	1	L	BRANCH OFFSET																								Branch
Condition				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				Branch Exchange
Condition				1	1	0	P	U	N	W	L	Rn				CRd				CPNum				OFFSET								COPROCESSOR DATA XFER
Condition				1	1	1	0	Op-1				CRn				CRd				CPNum				OP-2			0	CRm				COPROCESSOR DATA OP
Condition								OP-1			L	CRn				Rd				CPNum				OP-2			1	CRm				COPROCESSOR REG XFER
Condition				1	1	1	1	SWI NUMBER																								Software Interrupt

ADI SHARC DSP Instruction Set Encoding

Type 1: Compute | DregX«…DM | DregY«…PM

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	PD		DD		AMF					YOP		XOP			PMI		PMM		DMI		DMM		

Type 3: Dreg/Ireg/Mreg «…» DM/PM Register

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	D	16-bit address									16-bit address							IREG/MREG			

Type 4: Compute | Dreg «…» DM

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	G	D	Z	AMF					YOP		XOP			DREG				I		M	

Type 6: Dreg «… Data16

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	16-bit data									16-bit data							DREG			

Type 7: Reg1/2 «… Data16

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	16-bit data									16-bit data							REG1			

Type 8: Compute | Dreg1 «… Dreg2

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	Z	AMF					YOP		XOP			DDREG				SDREG			

Type 9: Compute

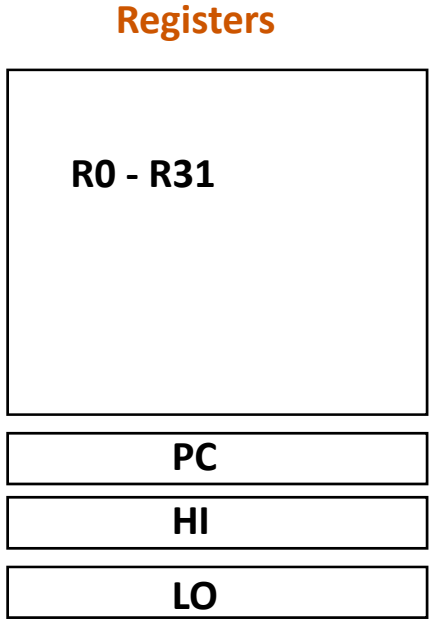
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Z	AMF					YOP		XOP			0	0	0	0	COND			

Type 10: Jump

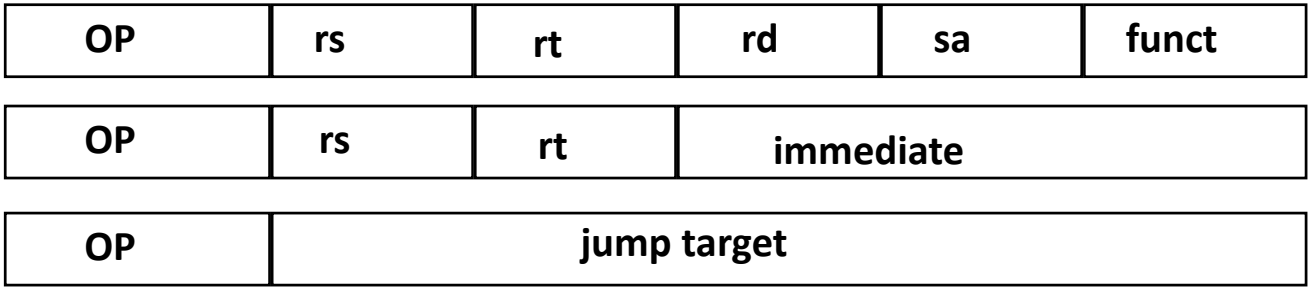
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	B	13-bit address						13-bit address							COND			

MIPS R3000 Instruction Set Encoding

- **Instruction Categories**
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



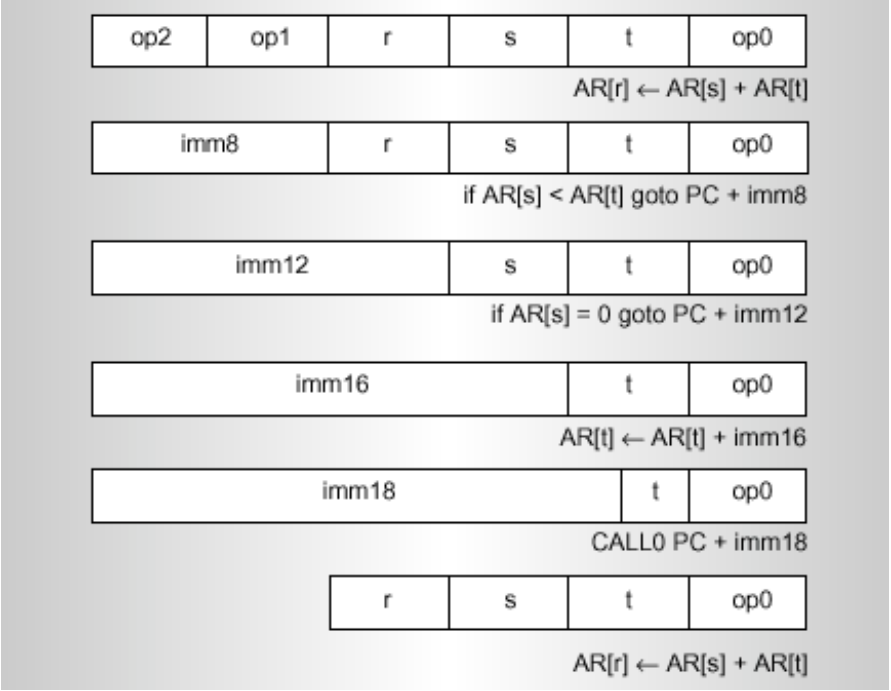
3 Instruction Formats: all 32 bits wide



Lee, SJSU.edu

Tensilica Instruction Set Encoding

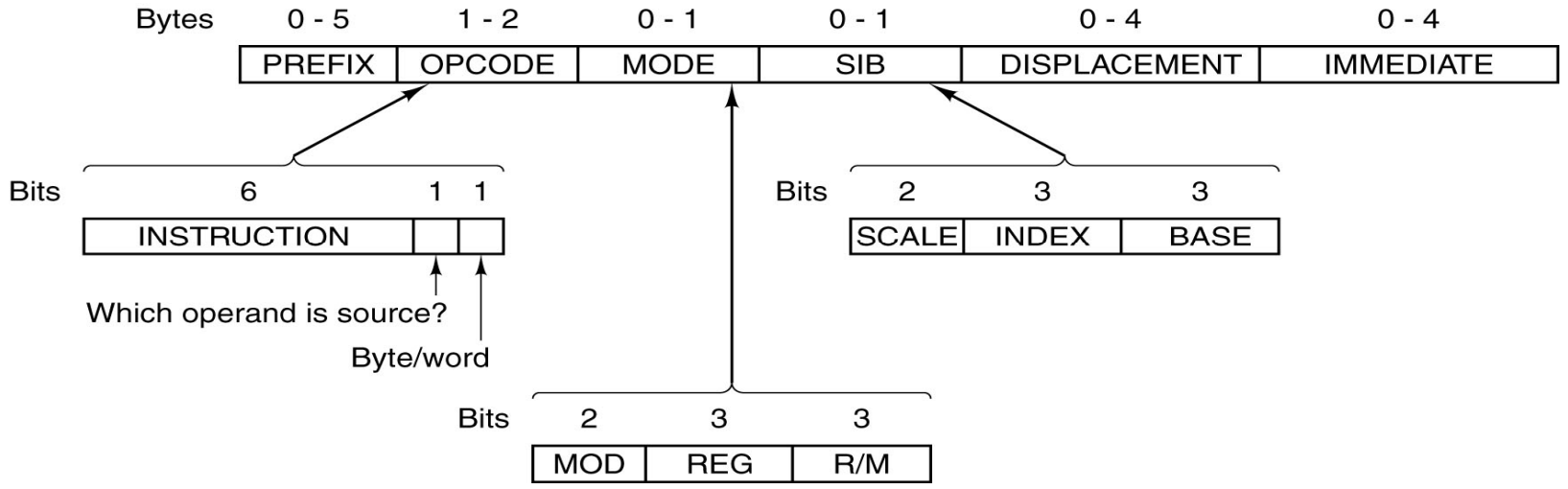
- ISA defines 24 and 16 bit instruction formats (as opposed to 32-bit)
  - Consists of:
    - 78 instructions
    - five-stage pipeline that supports single-cycle execution
    - 1 - load/store model
    - 32-entry orthogonal register file
    - 32 optional extra registers
- Attributes:
  - Enables configurability
  - Minimizes code size
  - Reduces power req.
  - Maximizes performance
- High performance gained by:
  - Richer instructions
  - Reducing register save and restore overhead in subroutine calls
  - Zero overhead loop instructions



Tensilica, Inc. Documentation

The Pentium 4 Instruction Set Encoding

- Variable length instructions with up to 6 variable length fields.
- The 8088 instructions all had 1 byte opcodes plus the option of a prefix byte.
- As the instruction set expanded, the opcodes were exhausted. The 0xFF opcode was used to indicate the use of the second opcode byte.
- Very complex decoding



Intel Corp.

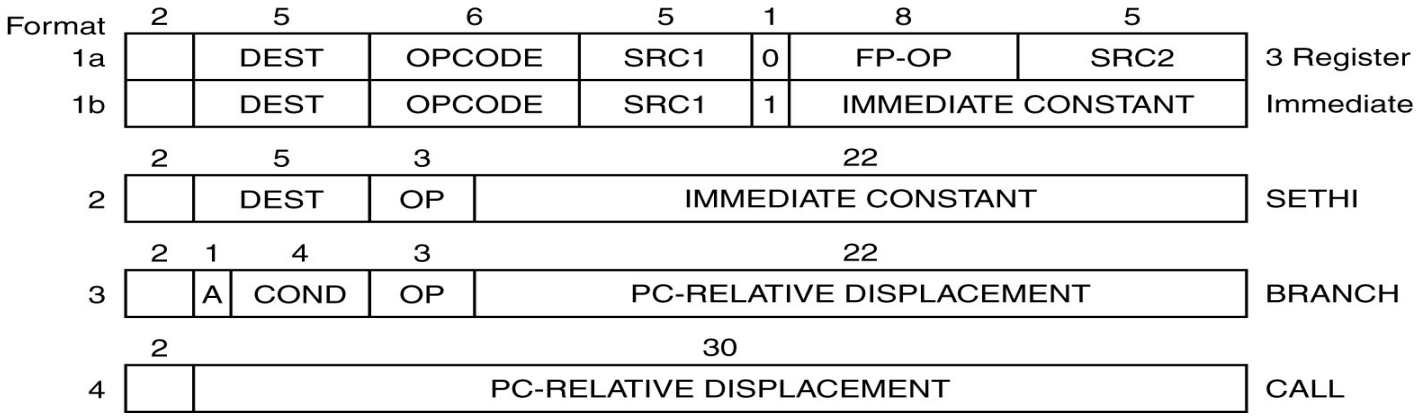
X86 Instruction Usage: Top 10

Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

Observation: Simple instructions dominate instruction usage frequency.

The UltraSPARC III Instruction Encoding

- 32-bit, fixed length instructions with expanding opcodes.
- Most instructions specify an operation and three registers, one of which is the destination for the result.
  - Originally, 5 formats, now 31.
  - the first 2 bits determine (for the most part) the instruction format.
- Format 1 handles instructions with two source registers, or a source register and a constant (12 bits).
- Format 2 is used to create a 32-bit constant in a register using a sequence of two instructions. The SETHI instruction sets the upper 22 bits.
- Format 3 provides branching capabilities using PC-relative addressing.
  - The 22-it displacement (signed) is shortened to 19-bits to accommodate the predictive branch instructions (stealing bits from the displacement to implement new instructions).
  - The CALL instruction has its own format. The opcode is implied by the first 2 bits.



SUN, Inc

Types of ISAs According To Operand Addressing Fields

- **Memory-To-Memory Machines:**
  - Operands obtained from memory and results stored back in memory by any instruction that requires operands.
  - No local CPU registers are used in the CPU datapath.
  - Include:
    - The 4 Address ISA.
    - The 3-Address ISA.
    - The 2-Address ISA.
- **The 1-address (Accumulator) ISA:**
  - A single local CPU special-purpose register (accumulator) is used as the source of one operand and as the result destination.

Types of ISAs According To Operand Addressing Fields

- **The 0-address or Stack ISA:**
  - A push-down stack is used in the CPU.
- **General Purpose Register (GPR) ISA:**
  - The CPU datapath contains several local general-purpose registers which can be used as operand sources and as result destinations.
  - A large number of possible addressing modes.
  - Load-Store or Register-To-Register Machines: GPR machines where only data movement instructions (loads, stores) can obtain operands from memory and store results to memory.

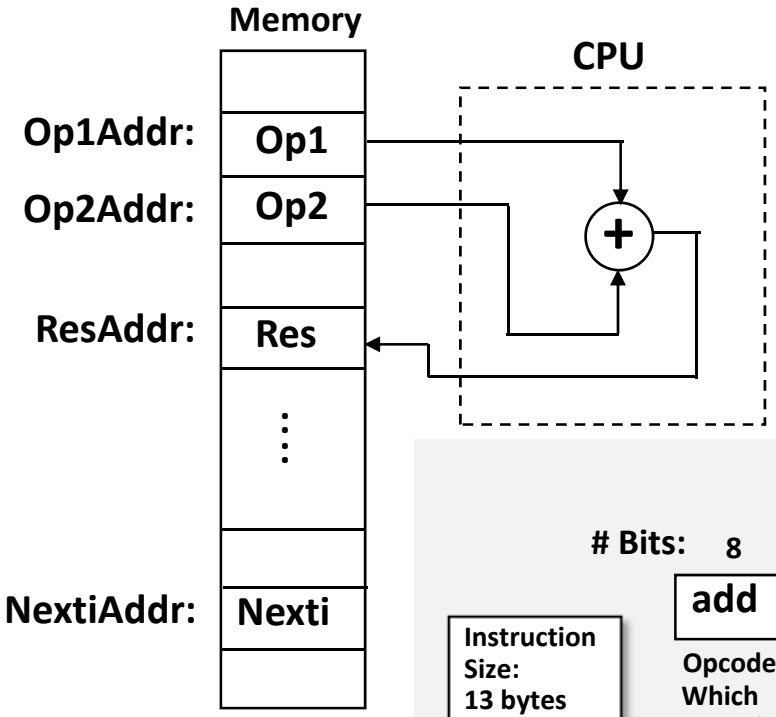
Types of ISAs - Memory-To-Memory Machines: The 4-Address ISA

No program counter (PC) or other CPU registers are used.

Instruction encoding has four address fields to specify:

Location of first operand. - Location of second operand.

Place to store the result. - Location of next instruction.



Instruction:

add Res, Op1, Op2, Nexti

Meaning:

Res <- Op1 + Op2

or more precise RTN:

M[ResAddr] <- M[Op1Addr] + M[Op2Addr]

Instruction Format (encoding)

# Bits:	8	24	24	24	24
	add	ResAddr	Op1Addr	Op2Addr	NextiAddr
	Opcode Which operation	Where to put result	Where to find operands		Where to find next instruction

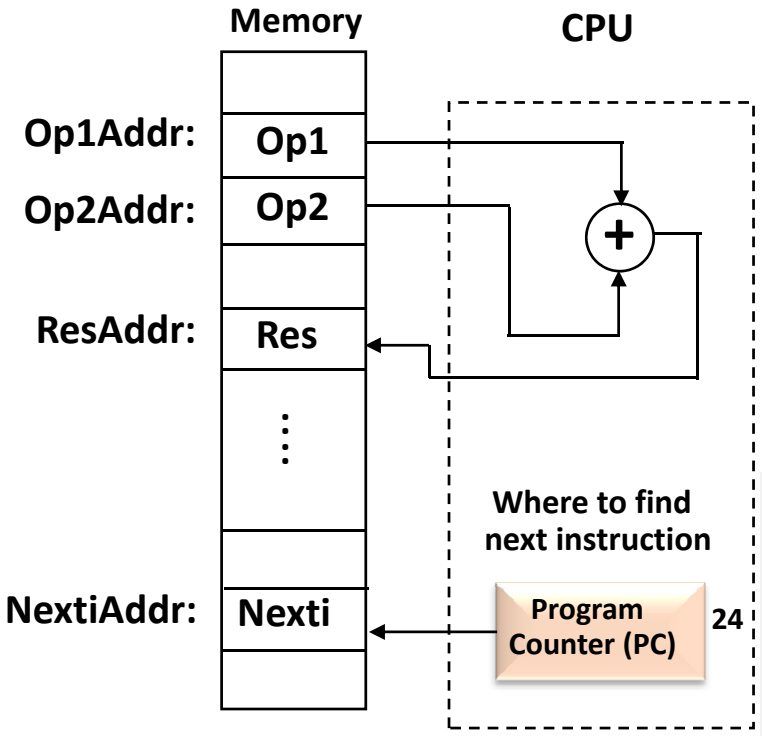
Instruction Size: 13 bytes

Can address 2^{24} bytes = 16 MBytes

Types of ISAs - Memory-To-Memory Machines: The 3-Address ISA

A program counter (PC) is included within the CPU which points to the next instruction.

No CPU storage (general-purpose registers).



Can address 2^{24} bytes = 16 MBytes

Instruction:

sub Res, Op1, Op2

Meaning:

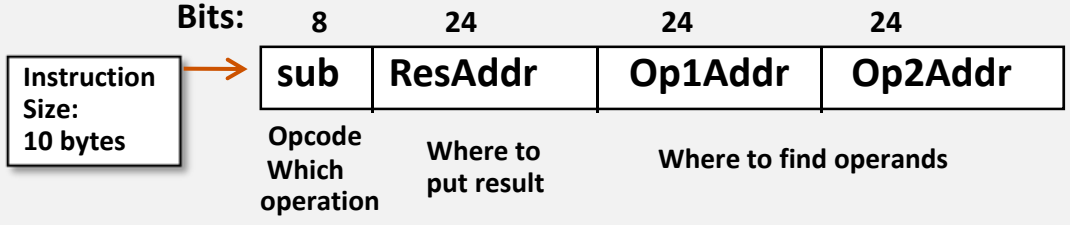
Res <- Op1 - Op2

or more precise RTN:

M[ResAddr] <- M[Op1Addr] - M[Op2Addr]

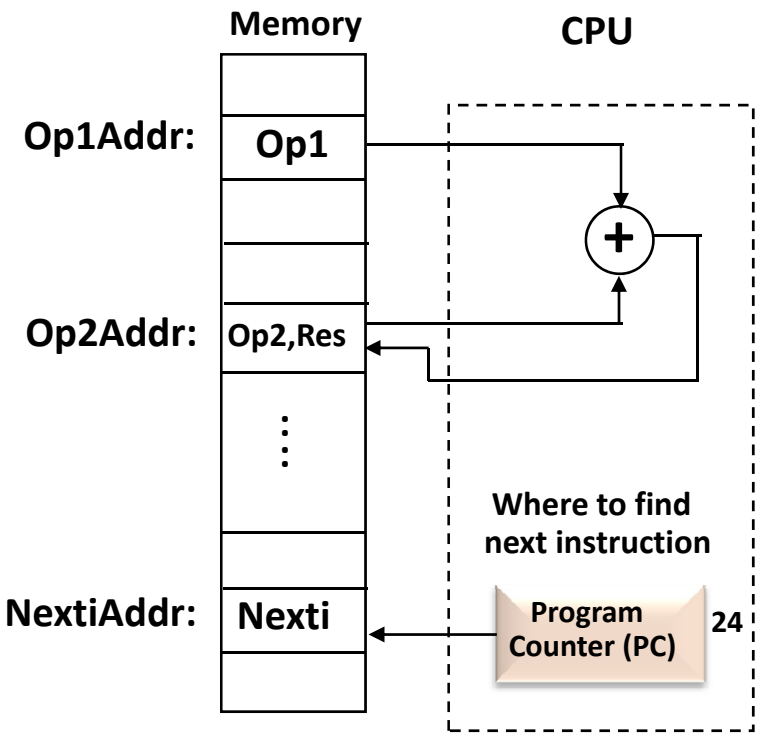
PC <- PC + 10 ← Increment PC

Instruction Format (encoding)



Types of ISAs - Memory-To-Memory Machines: The 2-Address ISA

The 2-address Machine: Result is stored in the memory address of one of the operands.



Can address 2^{24} bytes = 16 MBytes

Instruction:

add Op2, Op1

Meaning:

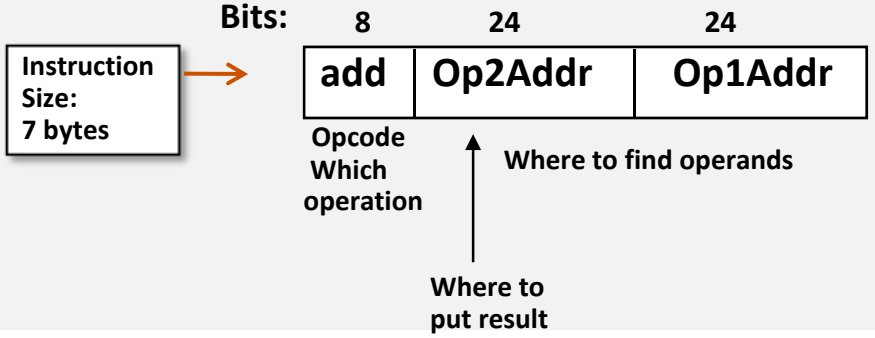
Op2 <- Op1 + Op2

or more precise RTN:

M[Op2Addr] <- M[Op1Addr] + M[Op2Addr]

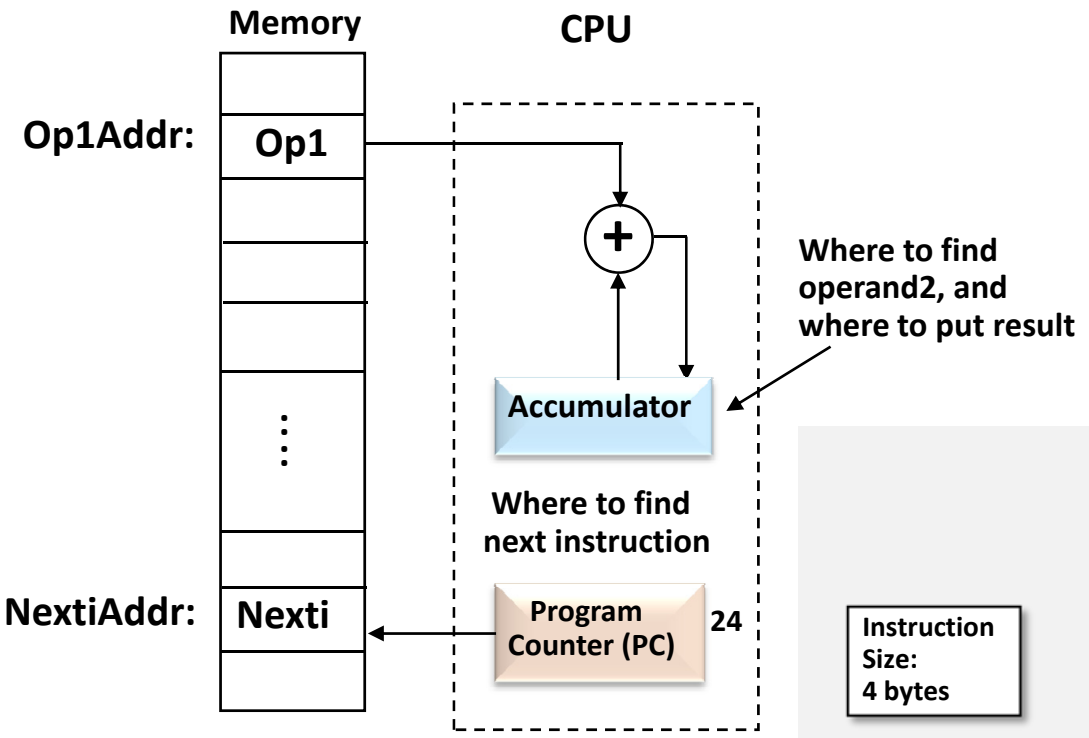
PC <- PC + 7 ← Increment PC

Instruction Format (encoding)



Types of Instruction Set Architectures
The 1-address (Accumulator) Machine

A single accumulator in the CPU is used as the source of one operand and result destination.



Instruction:

add Op1

Meaning:

Acc <- Acc + Op1

or more precise RTN:

Acc <- Acc + M[Op1Addr]
PC <- PC + 4

Increment PC

Instruction Format (encoding)

Instruction Size: 4 bytes

Bits: 8 | 24

add	Op1Addr

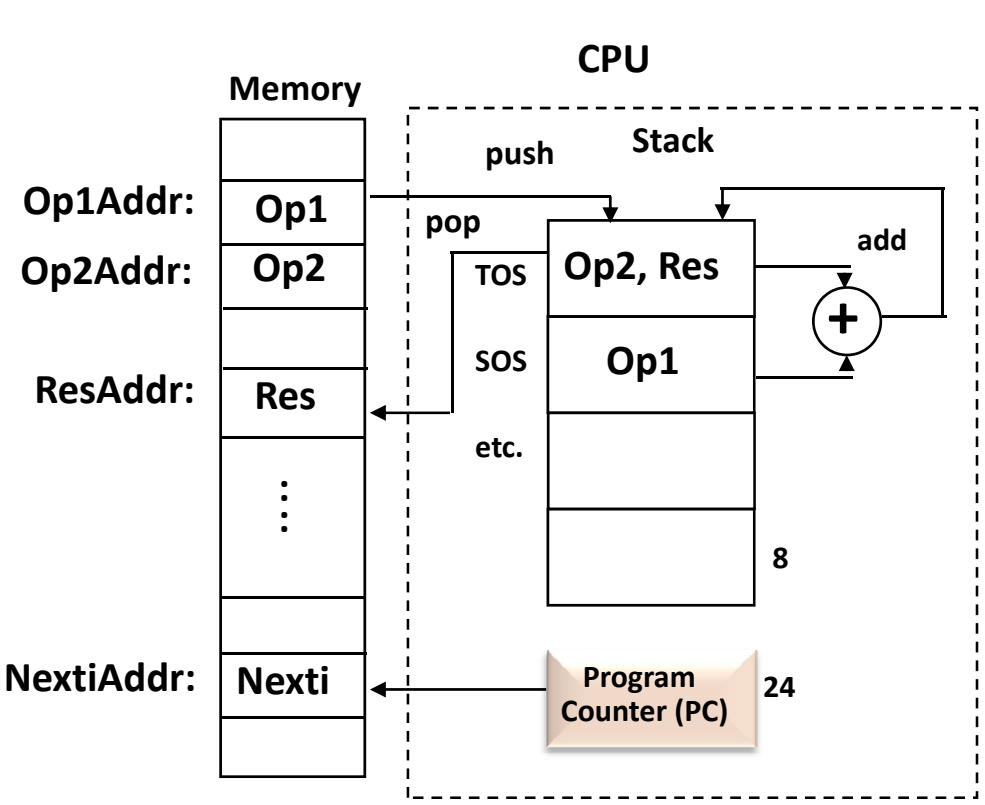
Opcode Which operation

Where to find operand1

Can address 2^{24} bytes = 16 MBytes

Types of Instruction Set Architectures
The 0-address (Stack) Machine

A push-down stack is used in the CPU.



TOS = Top Entry in Stack
SOS = Second Entry in Stack

4 Bytes — Instruction Format

Bits: 8 | 24

push	Op1Addr
Opcode	Where to find operand

Instruction: push Op1

Meaning: TOS <- M[Op1Addr]

Instruction: add

1 Byte — Instruction Format

Bits: 8

add
Opcode

Meaning: TOS <- TOS + SOS

4 Bytes — Instruction Format

Bits: 8 | 24

pop	ResAddr
Opcode	Memory Destination

Instruction: pop Res

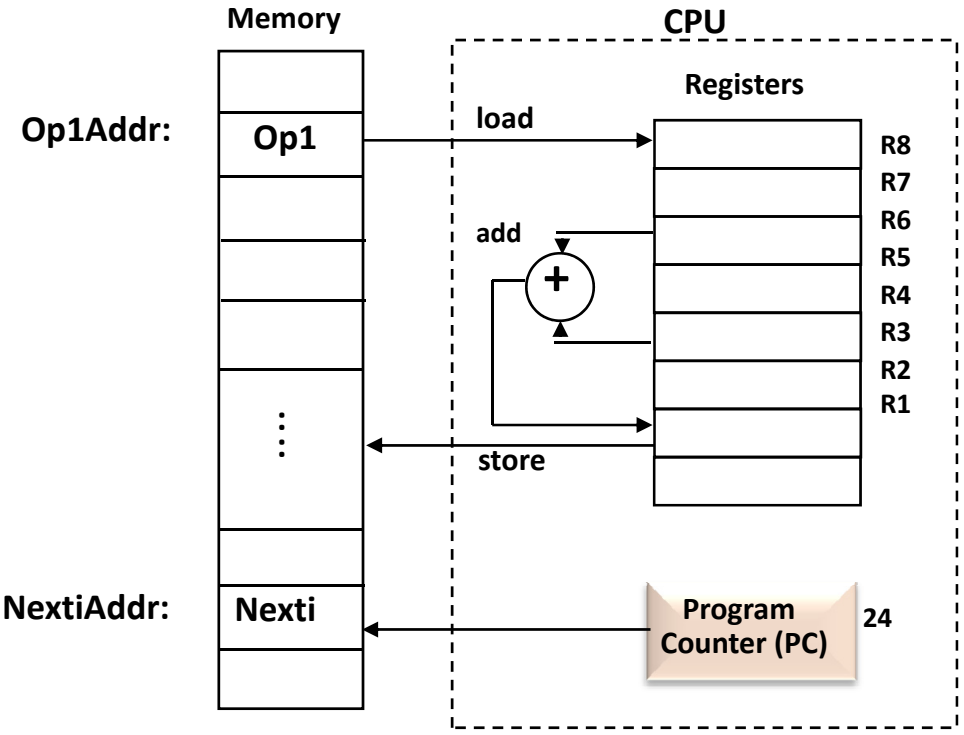
Meaning: M[ResAddr] <- TOS

Can address 2^{24} bytes = 16 MBytes

Types of Instruction Set Architectures
General Purpose Register (GPR) Machines

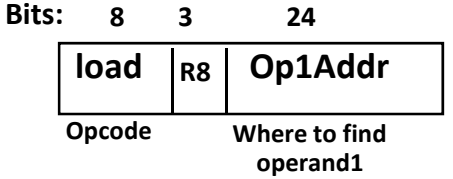
CPU contains several general-purpose registers which can be used as operand sources and result destination.

Eight general purpose Registers (GPRs) assumed here: R1-R8



Instruction:
load R8, Op1
Meaning:
R8 <- M[Op1Addr]
PC <- PC + 5

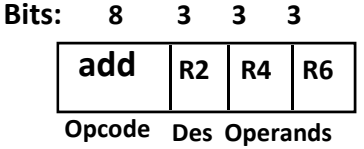
Instruction Format



Size = 4.375 bytes rounded up to 5 bytes

Instruction:
add R2, R4, R6
Meaning:
R2 <- R4 + R6
PC <- PC + 3

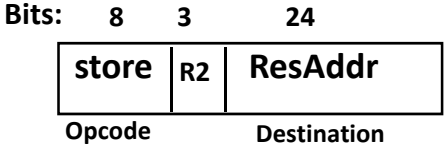
Instruction Format



Size = 2.125 bytes rounded up to 3 bytes

Instruction:
store R2, Op2
Meaning:
M[Op2Addr] <- R2
PC <- PC + 5

Instruction Format



Size = 4.375 bytes rounded up to 5 bytes

The add instruction has three register specifier fields
While load, store instructions have one register specifier field and one memory address specifier field

Expression Evaluation Example with 3-, 2-, 1-, 0-Address, And GPR Machines

For the expression A = (B + C) * D - E where A-E are in memory

3-Address	2-Address	1-Address Accumulator	0-Address Stack	GPR	
				Register-Memory	Load-Store

add A, B, C
mul A, A, D
sub A, A, E
	load A, B
	add A, C
	mul A, D
	sub A, E
		load B
		add C
		mul D
		sub E
		store A
			push B
			push C
			add
			push D
			mul
			push E
			sub
			pop A
				load R1, B
				add R1, C
				mul R1, D
				sub R1, E
				store A, R1
					load R1, B
					load R2, C
					add R3, R1, R2
					load R1, D
					mul R3, R3, R1
					load R1, E
					sub R3, R3, R1
					store A, R3

3 instructions Code size: 30 bytes 9 memory accesses for data	4 instructions Code size: 28 bytes 11 memory accesses for data	5 instructions Code size: 20 bytes 5 memory accesses for data	8 instructions Code size: 23 bytes 5 memory accesses for data	5 instructions Code size: 25 bytes 5 memory accesses for data	8 instructions Code size: 34 bytes 5 memory accesses for data

Instruction Set Architecture Tradeoffs

- 3-address machine: shortest code sequence; a large number of bits per instruction; large number of memory accesses.
- 0-address (stack) machine: Longest code sequence; shortest individual instructions; more complex to program.
- General purpose register machine (GPR):
  - Addressing modified by specifying among a small set of registers with using a short register address (all new ISAs since 1975).
  - Advantages of GPR:
    - Low number of memory accesses. Faster, since register access is currently still much faster than memory access.
    - Registers are easier for compilers to use.
    - Shorter, simpler instructions.
- Load-Store Machines: GPR machines where memory addresses are only included in data movement instructions (loads/stores) between memory and registers.

Instruction Set Architecture Tradeoffs (cont)

- **Accumulator Machines**
  - Requires storing lots of temporary and intermediate values in memory
  - Accumulator is only really beneficial for a chain (sequence) of calculations where the result of one is the input to the next.
- **Memory-to-Memory ISAs**
  - Main memory much slower than arithmetic circuits
    - This was as true in 1950 as in 2009!
  - It takes a lot of room to specify memory addresses
  - Results are often used one or two instructions later

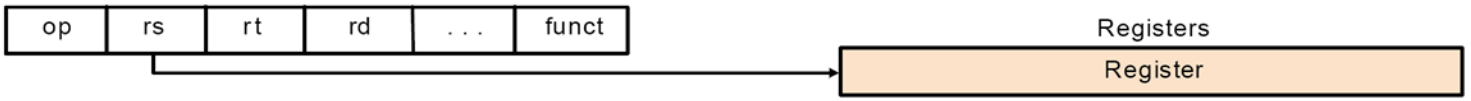
Typical GPR ISA Memory Addressing Modes

Addressing Mode	Sample Instruction	Meaning
Register	add R4, R3	R4 <- R4 + R3
Immediate	add R4, #3	R4 <- R4 + 3
Displacement	add R4, 10 (R1)	R4 <- R4 + Mem[10+ R1]
Indirect	add R4, (R1)	R4 <- R4 + Mem[R1]
Indexed	add R3, (R1 + R2)	R3 <- R3 +Mem[R1 + R2]
Absolute	add R1, (1001)	R1 <- R1 + Mem[1001]
Memory indirect	add R1, @ (R3)	R1 <- R1 + Mem[Mem[R3]]
Autoincrement	add R1, (R2) +	R1 <- R1 + Mem[R2] R2 <- R2 + d
Autodecrement	add R1, - (R2)	R2 <- R2 - d R1 <- R1 + Mem[R2]
Scaled	add R1, 100 (R2) [R3]	R1 <- R1+ Mem[100+ R2 + R3*d]

NOTE: "d" is generally the size of the memory word width e.g. 4 bytes.

Addressing Modes

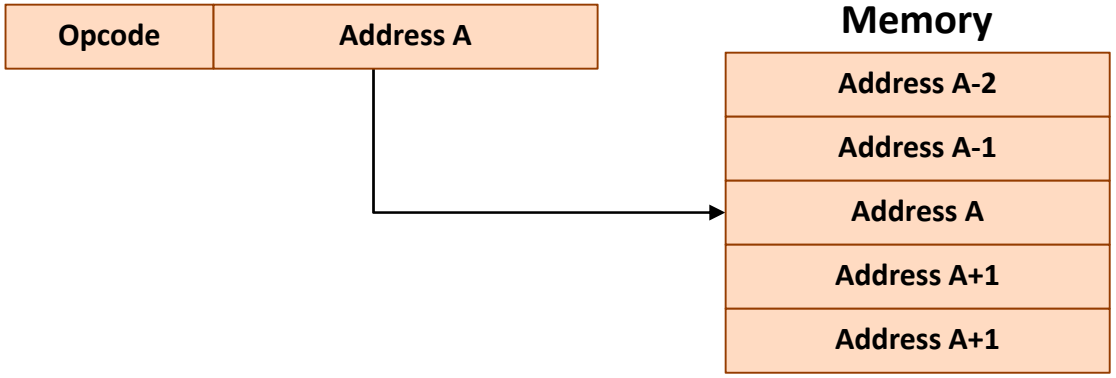
- **Register addressing.**
  - Specify register number rather than address.
    MOV R1, R2
  - Copy content of register 2 to register 1.



Addressing Modes

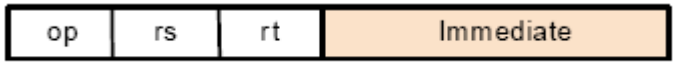
- **Direct addressing:**
  - Give full address of operand in memory.

    MOV R1, #A
  - Load word from address of static variable A to register 1.
  - Single memory reference to access data
  - No additional calculations to work out effective address
  - Limited address space



Addressing Modes

- **Immediate addressing:**
  - Address part of operand contains operand itself.
    - MOV R1, #4
    - Load constant 4 to register 1.
  - Only small integer constants can be specified in this way.
  - No memory reference to fetch data
  - Fast
  - Limited range



Addressing Modes

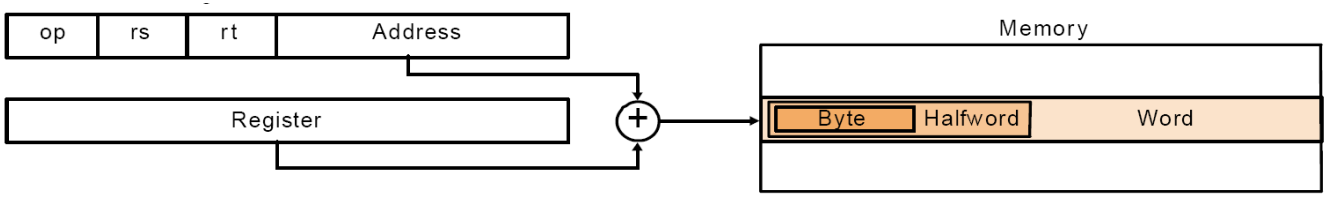
- **Register Indirect Addressing**
  - Operand address is not contained in instruction but in a register.
  - Operand address is a pointer.
    ADD R1, (R2); add to register R1 word at address contained in R2.
  - Can refer to different addresses in different instruction.
    - Example: assembly code for adding the elements of an array.

```
        MOV R1, #0       ; accumulate sum in R1, initially 0
        MOV R2, #A       ; R2 = address of the array A
        MOV R3, #A+4096  ; R3 = address of first word beyond A
LOOP:   ADD R1, (R2)     ; register indirect through R2 to get operand
        ADD R2, #4       ; increment R2 by one word (4 bytes)
        CMP R2, R3       ; are we done yet?
        BLT LOOP         ; if R2 < R3, we are not done, so continue
```

Addressing Modes

- **Indexed Addressing**
  - Memory is addressed by giving a register plus a constant offset.
    - Example: processing of static arrays.

      MOV R4, A(R2); load into R1 word whose address has offset A from content of R2.



  - Array is at a fixed address; register contains current index.
    - Example: assembly code for computing $\Sigma_i A_i * B_i$.

```
        MOV R1, #0     ; accumulate the sum in R1, initially 0
        MOV R2, #0     ; R2 = index i
        MOV R3, #4096  ; R3 = first index value not in use
LOOP:   MOV R4, A(R2)  ; R4 = A[i]
        MUL R4, B(R2)  ; R4 = A[i] * B[i]
        ADD R1, R4     ; sum all the products into R1
        ADD R2, #4     ; i = i+4 (1 word = 4 bytes)
        CMP R2, R3     ; are we done yet?
        BLT LOOP       ; if R2 < R3, we are not done, so continue
```

Addressing Modes

- **Based-Indexed Addressing**
  - Address is computed by sum of two registers plus optional offset.
  - Processing of dynamic arrays.
    MOV R4, (R2+R5): load into R4 word whose address is the sum of R2 and R5.
  - R5 is the base address of the array.
  - R2 is the current index.
  - Replace loop code in previous example as follows:

```
        ...
        MOV R5, #A       ; R5 = address of A
        MOV R6, #B       ; R6 = address of B
LOOP:   MOV R4, (R2+R5)  ; R4 = A[i]
        MUL R4, (R2+R6)  ; R4 = A[i] * B[i]
        ...
```

Addressing Modes

- **Memory Indirect**
  - Memory cell pointed to by address field contains the address of (pointer to) the operand

    EA = (A)

    Look in A, find address (A) and look there for operand
  - e.g. ADD (A)
    - Add contents of cell pointed to by contents of A to accumulator
  - Large address space
  - 2^n where n = word length
  - May be nested, multilevel, cascaded
    - e.g. EA = (((A)))
  - Multiple memory accesses to find operand
  - Very slow

Addressing Modes

- **Stack Addressing**
  - Zero-address instructions use stack to avoid explicit memory addresses.
  - Example: code for evaluation of (8 + 2 x 5)/(1 + 3 x 2 - 4)
  - Reverse Polish notation: 8 2 5 x + 1 3 2 x + 4 -

Step	Remaining String	Instruction	Stack
1	$8\ 2\ 5\ \times\ +\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$	BIPUSH 8	8
2	$2\ 5\ \times\ +\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$	BIPUSH 2	8, 2
3	$5\ \times\ +\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$	BIPUSH 5	8, 2, 5
4	$\times\ +\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$	IMUL	8, 10
5	$+\ 1\ 3\ 2\ \times\ +\ 4\ -\ /$	IADD	18
6	$13\ 2\ \times\ +\ 4\ -\ /$	BIPUSH 1	18, 1
7	$3\ 2\ \times\ +\ 4\ -\ /$	BIPUSH 3	18, 1, 3
8	$2\ \times\ +\ 4\ -\ /$	BIPUSH 2	18, 1, 3, 2
9	$\times\ +\ 4\ -\ /$	IMUL	18, 1, 6
10	$+\ 4\ -\ /$	IADD	18, 7
11	$4\ -\ /$	BIPUSH 4	18, 7, 4
12	$-/$	ISUB	18, 3
13	$/$	IDIV	6

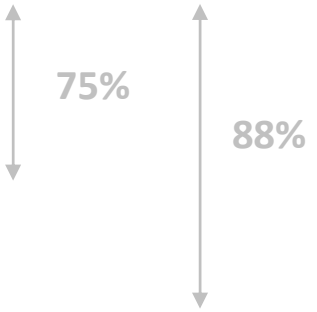
Addressing Modes for Branch Instructions

- How to specify target address of branch instructions/procedure calls?
  - Direct addressing: unconditional branches (gotos).
    - Generated from conditionals and loops.
  - Register indirect addressing or indexed mode.
    - Program may compute target address (computed goto, switch).
  - PC-relative addressing: indexed mode where PC acts as register.
    - Target address is specified as offset to current instruction.
- Modes presented so far are also useful for branch instructions

VAX 11/780 Addressing Modes Usage Example

For 3 programs running on VAX ignoring direct register mode:

Displacement	42% avg, 32% to 55%
Immediate:	33% avg, 17% to 43%
Register deferred (indirect):	13% avg, 3% to 24%
Scaled:	7% avg, 0% to 16%
Memory indirect:	3% avg, 1% to 6%
Misc:	2% avg, 0% to 3%



75% displacement & immediate
88% displacement, immediate & register indirect.

Observation: In addition Register direct, Displacement, Immediate, Register Indirect addressing modes are important.

Considerations in designing an ISA

- **Architectural State: Memory and Registers**
  - Abstraction used by both compiler and Microarchitecture
  - Register specialization?
  - RISC and VLIW tend to use a large pool of general-purpose registers
  - Implementation techniques provide illusion of large number of registers or fast flat memory space (register renaming)

- **Pipelining and operational latency**
  - Sequential execution v.s pipelined execution model
  - Exposed or hidden pipelining (latency of operations): delayed branches, bypass networks, scoreboard
  - DSPs enforce uniform latency model

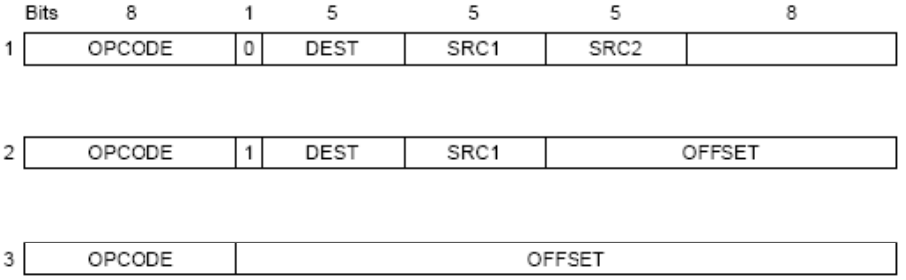
- **Delayed branches expose part of the RISC pipeline**

Considerations in designing an ISA

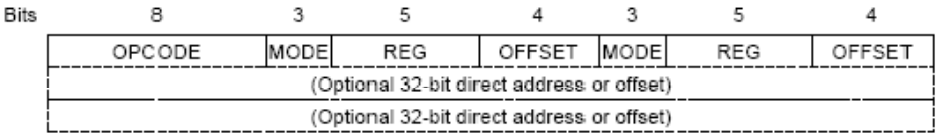
- **Encoding and Architectural Style:**
  - Architecture description contains architectural state, instruction set, and execution model
    - Implementations are free to build whatever structure they want, as long as they simulate the architectural model
  - RISC Encodings
    - Single instruction corresponds to a single operation
    - 1:1 encoding (Instruction encoding size: Number of run-time operations
  - CISC Encodings
    - RISC subset plus more complex instructions
    - Variable: Dynamic encoding
  - VLIW Encodings
    - n:n ; Fallacy: VLIW Instructions Have Fixed Width
  - DSP Encodings
    - 1:n
  - Vector Encodings
    - 1:variable

Considerations in designing an ISA

- In a clean design, every opcode should permit every addressing mode.
  - Three Address Machine
    - Two formats selected by bit.
    - 1 special format for branches



  - Two Address Machine
    - Each operand specified by 12 bits.
    - Mode, register, offset.
    - Optional 32-bit word for address.



Considerations in designing an ISA

- **Multiple Issues and Hazards**
  - Exposing Dependence (DEP) and Independence (INDEP); Resource hazards

	Sequential Architectures	Dependence Architectures	Independence Architectures
Example	Superscalar	Dataflow	VLIW
DEP info in program?	Implicit in registers	Exact description	Descr. of independent ops
How DEPops exposed?	By hardware	By compiler	By compiler
How INDEPops exposed?	By hardware	By hardware	By compiler
Schedule?	By hardware	By hardware	By compiler
Compiler	Rearranges code	Some data dep analysis	Everything

Considerations in designing an ISA

- **Exceptions and Interrupts**
  - Precise and Imprecise exceptions
  - Fast & Slow Interrupts
- **Trade-offs in exposing or hiding implementation features**
  - Execution latency
  - Out-of-order execution
  - Bypass networks
  - Threading
  - Architectural state

ISA Design Example: VLIW ISA Design Principles

- **VLIW instructions consist of parallel operations**
  - These operations execute in parallel, on parallel hardware
  - Compiler guarantees parallel execution is safe and correct
- **Compiling to target a specific implementation is required for correctness**
  - Operation latency, functional unit organization, and clustering is visible to the compiler
  - Recompilation necessary on new architecture
- **Implementations execute parallelism noted by the compiler**
  - Implementation trusts the compiler to produce fast and correct code
  - No hardware to check for structural and resource hazards
- **Expose power, and hide weakness**
  - Some features not worth exposing: bypass networks, clean interrupt handling

ISA Design Example: VLIW Execution Model Subtleties

- **Horizontal issues – within the instruction word**
  - Two operations read and write the same register
    - The read sees the original value of the register
    - The read sees the value written by the write
    - Illegal combination

mov $r1=1	mov $r2=2

mov $r1=$r2	mov $r2=$r1

  - How operations within instructions are completed in the case of exception?
    - "none complete"
    - "all that can complete"
    - "all operations before the excepting operation complete"

- **Vertical issues – across pipelined instructions**
  - How pipeline latencies are exposed:
    Whether an operation has a fixed latency for all the cases or whether it is allowed to finish in some cases earlier: EQ model and LEQ model

Summary

- Programmable CPU cores are important for the control parts of typical embedded application.
- They are well supported with tools to support the development of end-user software. (vs. deeply embedded SW)
- "Keep it Simple" heuristic (RISC vs. CISC)
  - Make frequent cases fast and rare cases correct.
  - Regular (orthogonal) instruction set
  - No special features that match a high level language construct.
  - At least 16 registers to ease register allocation.
- Embedded cores are often light cores which are a compromise between performance, area and power dissipation.
  (vs. stand-alone CPU cores which are optimized for performance)

Backup

Motorola 680X0

18 addressing modes:

- Data register direct.
- Address register direct.
- Immediate.
- Absolute short.
- Absolute long.
- Address register indirect.
- Address register indirect with postincrement.
- Address register indirect with predecrement.
- Address register indirect with displacement.
- Address register indirect with index (8-bit).
- Address register indirect with index (base).
- Memory inderect postindexed.
- Memory indirect preindexed.
- Program counter indirect with index (8-bit).
- Program counter indirect with index (base).
- Program counter indirect with displacement.
- Program counter memory indirect postindexed.
- Program counter memory indirect preindexed.

GPR ISA (Register-Memory)

Operand size:

- Range from 1 to 32 bits, 1, 2, 4, 8, 10, or 16 bytes.

Instruction Encoding:

- Instructions are stored in 16-bit words.
- the smallest instruction is 2-bytes (one word).
- The longest instruction is 5 words (10 bytes) in length.

Intel 80386

12 addressing modes:

- Register.
- Immediate.
- Direct.
- Base.
- Base + Displacement.
- Index + Displacement.
- Scaled Index + Displacement.
- Based Index.
- Based Scaled Index.
- Based Index + Displacement.
- Based Scaled Index + Displacement.
- Relative.

X86 or IA-32

GPR ISA (Register-Memory)

Operand sizes:

- Can be 8, 16, 32, 48, 64, or 80 bits long.
- Also supports string operations.

Instruction Encoding:

- The smallest instruction is one byte.
- The longest instruction is 12 bytes long.
- The first bytes generally contain the opcode, mode specifiers, and register fields.
- The remainder bytes are for address displacement and immediate data.

PowerPC

8 addressing modes:

- Register direct.
- Immediate.
- Register indirect.
- Register indirect with immediate index (loads and stores).
- Register indirect with register index (loads and stores).
- Absolute (jumps).
- Link register indirect (calls).
- Count register indirect (branches).

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 15 different formats with many minor variations.
- All are 32 bits in length.

SPARC

5 addressing modes:

- Register indirect with immediate displacement.
- Register inderect indexed by another register.
- Register direct.
- Immediate.
- PC relative.

Operand sizes:

- Four operand sizes: 1, 2, 4 or 8 bytes.

Instruction Encoding:

- Instruction set has 3 basic instruction formats with 3 minor variations.
- All are 32 bits in length.