Instruction Set Architecture (ISA)

ISA LEVEL

ELEMENTS OF INSTRUCTIONS

INSTRUCTIONS TYPES

INSTRUCTIONS FORMAT

Instruction Types: Categories

- Data processing
  - Arithmetic and logic instructions
- Data storage (main memory)
  - Memory instructions
    - movement of data into or out of memory locations
- Data movement (I/O)
  - I/O instructions
- Control (Program flow control)
  - Test and branch instructions

Instruction Types:
Examples of Data Movement and Data Storage

(i/o instructions & mem instructions)

Table 10.8 x86 Operation Types (with Examples of Typical Operations)

Instruction	Description
Data Movement	
MOV	Move operand, between registers or between register and memory.
PUSH	Push operand onto stack.
PUSHA	Push all registers on stack.
MOVSX	Move byte, word, dword, sign extended. Moves a byte to a word or a word to a doubleword with twos-complement sign extension.
LEA	Load effective address. Loads the offset of the source operand, rather than its value to the destination operand.
XLAT	Table lookup translation. Replaces a byte in AL with a byte from a user-coded translation table. When XLAT is executed, AL should have an unsigned index to the table. XLAT changes the contents of AL from the table index to the table entry.
IN, OUT	Input, output operand from I/O space.

Instruction Types: Examples of Data Processing

(arithmetic & logic instructions)

Table 10.8 x86 Operation Types (with Examples of Typical Operations)

Instruction	Description
Arithmetic	
ADD	Add operands.
SUB	Subtract operands.
MUL	Unsigned integer multiplication, with byte, word, or double word operands, and word, doubleword, or quadword result.
IDIV	Signed divide.
Logical	
AND	AND operands.
BTS	Bit test and set. Operates on a bit field operand. The instruction copies the current value of a bit to flag CF and sets the original bit to 1.
BSF	Bit scan forward. Scans a word or doubleword for a 1-bit and stores the number of the first 1-bit into a register.
SHL/SHR	Shift logical left or right.
SAL/SAR	Shift arithmetic left or right.

Instruction Types: Examples of Control

(test & branch instructions)

Table 10.8 x86 Operation Types (with Examples of Typical Operations)

Instruction	Description
Control Transfer	
JMP	Unconditional jump.
CALL	Transfer control to another location. Before transfer, the address of the instruction following the CALL is placed on the stack.
JE/JZ	Jump if equal/zero.
LOOPE/LOOPZ	Loops if equal/zero. This is a conditional jump using a value stored in register ECX. The instruction first decrements ECX before testing ECX for the branch condition.
INT/INTO	Interrupt/Interrupt if overflow. Transfer control to an interrupt service routine.

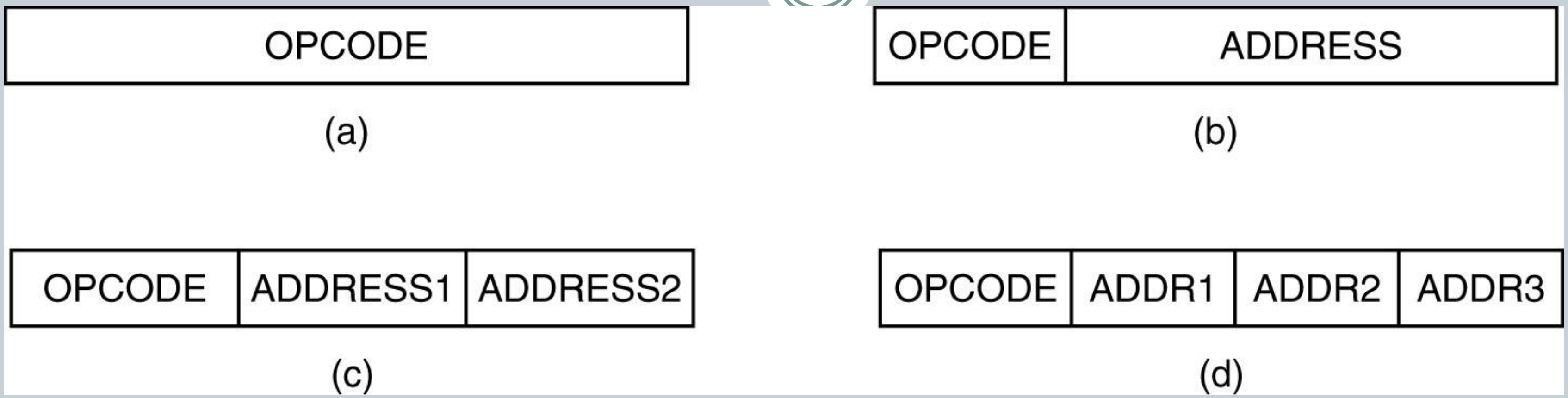
Instruction Set Architecture (ISA)

ISA LEVEL
ELEMENTS OF INSTRUCTIONS
INSTRUCTIONS TYPES
INSTRUCTIONS FORMAT

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Formats



Four common instruction formats:

(a) Zero-address instruction. (b) One-address instruction

(c) Two-address instruction. (d) Three-address instruction.

PDP-8 Instruction Format

Memory Reference Instructions

Opcode			D/I	Z/C	Displacement	
0		2	3	4	5	11

Input/Output Instructions

1	1	0	Device		Opcode	
0		2	3	8	9	11

Register Reference Instructions

Group 1 Microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 Microinstructions

1	1	1	1	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 Microinstructions

1	1	1	1	CLA	MQ
0	1	2	3	4	

D/I = Direct/Indirect address
Z/C = Page 0 or Current page
CLA = Clear Accumulator
CLL = Clear Link
CMA = CoMplement Accumulator
CML = CoMplement Link
RAR = Rotate Accumultator Right
RAL = Rotate Accumulator Left
BSW = Byte SWap

MQL = Multiplier Quotient Load

- One of the simplest instruction designs for a general-purpose computer was for.
- Fixed length instruction format.
- The PDP-8 uses 12-bit instructions and operates on 12-bit words.
- There is a single general-purpose register, the accumulator.

PDP-10 Instruction Format

Opcode	Register	I	Index Register	Memory Address
0 – 8	9 – 12		14 – 17	18 – 35

I = indirect bit

- The PDP-10 has a 36-bit word length and a 36-bit instruction length.
- Fixed length instruction format.

PDP-11 Instruction Format

1

Opcode	Source	Destination
4	6	6

2

Opcode	R	Source
7	3	6

3

Opcode	Offet
8	8

4

Opcode	FP	Destination
8	2	6

5

Opcode	Destination
10	6

6

Opcode	CC
12	4

7

Opcode	R
13	3

8

Opcode
16

9

Opcode	Source	Destination	Memory Address
4	6	6	16

10

Opcode	R	Source	Memory Address
7	3	6	16

11

Opcode	FP	Source	Memory Address
8	2	6	16

12

Opcode	Destination	Memory Address
10	6	16

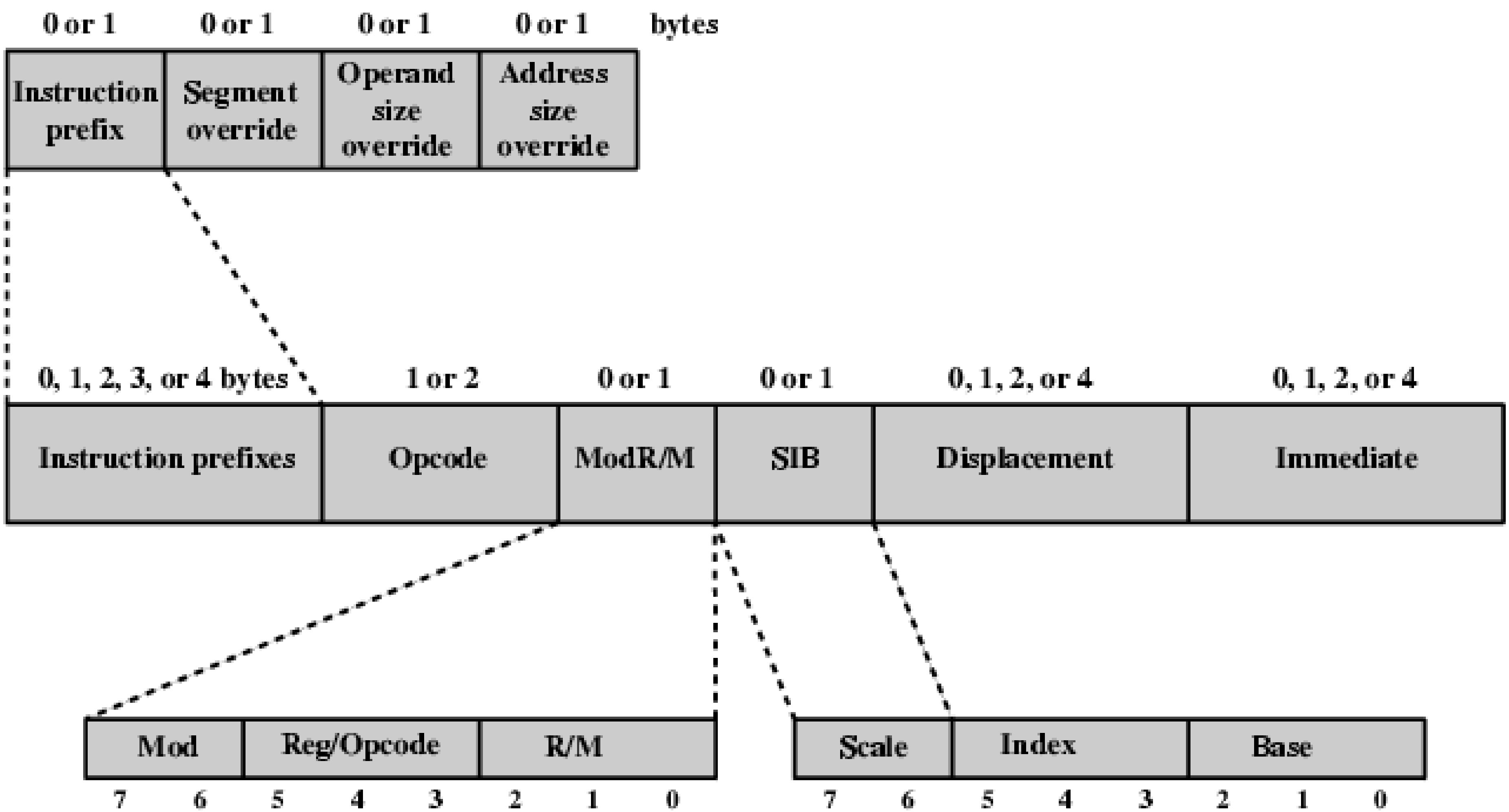
13

Opcode	Source	Destination	Memory Address 1	Memory Address 2
4	6	6	16	16

- PDP-11 was designed to provide a powerful and flexible instruction set within the constraints of a 16-bit minicomputer.
- Variable length instruction format.
- PDP-11 employs a set of eight 16-bit general-purpose registers.

Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

x86 Instruction Format



x86 Instruction Format

Opcode	ModR/M	SIB	displacement	Immediate
1 or 2 bytes	1 byte, If required	1 byte, If required	1,2 or 4 bytes If required	1,2 or 4 bytes If required

- Opcode: determine the action
- ModR/M: Addressing modes register/memory
- SIB: Scale-Index-Base
- Not all fields are present in all instr.
- If present, must be in the above order

x86 Instruction Format

ModR/M

Mod	Reg #	R/M
2 bits	3 bits	3 bits

- **Mod=00,**
  - First operand a register, specified by Reg #
  - Second operand in memory; address stored in a register numbered by R/M.
    - That is, Memory[Reg[R/M]]
  - Exceptions:
    - R/M=100 (SP): SIB needed
    - R/M=101 (BP): disp32 needed

x86 Instruction Format

Assembly Name	Reg #
EAX	000
EBX	001
ECX	010
EDX	011
ESP	100
EDP	101
ESI	110
EDI	111

x86 Instruction Format

- **Mod=01**, same as Mod 00 with 8-bit displacement.
  - Second operand: Memory[disp8+Reg[R/M].
  - Exception: SIB needed when R/M=100
- **Mod=10**, same as Mod 01 with 32-bit displacement
- **Mod=11**
  - Second operand is also a register, numbered by R/M.

x86 Instruction Format

SIB

Scale	Index	Base
2 bits	3 bits	3 bits

- Specify how a memory address is calculated
- Address=Reg[base] + Reg[Index]*2^{scale}
- Exceptions:
  - SP cannot be an index, and
  - BP cannot be a base.

x86 Instruction Format

Example: Add Instructions

- The first operand is the destination.
  - Can be register or memory
- The second operand is the source
  - Can be register or memory
- The two operands cannot be both memory.
- Action: dest += source

x86 Instruction Format

04	immd8		AL += immd8
05	immd32		EAX += immd32
00	modRM		Rm8 += r8
01	modRM		Rm32 += r32
03	modRM		r32 += rm32
80	11 000	immd8	Rm8 += immd8
81	11 000	immd32	Rm32 += immd32

x86: **add** instruction

Opcode	Mnemonic	Operand(s)
00	add	mem8,reg8
01	add	mem16,reg16
		mem32,reg32
02	add	reg8,reg8
02	add	reg8,mem8
03	add	reg16,reg16
		reg32,reg32
03	add	reg16,mem16
		reg32,mem32

x86: **add** instruction

Opcode	Mnemonic	Operand(s)
04	add	AL,imm8
05	add	AX,imm16
		EAX,imm32
80	add	reg8,imm8
80	add	mem8,imm8

ARM Instruction Formats

	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
data processing immediate shift	cond	0	0	0	opcode				S	Rn	Rd	shift amount		shift	0	Rm
data processing register shift	cond	0	0	0	opcode				S	Rn	Rd	Rs	0	shift	1	Rm
data processing immediate	cond	0	0	1	opcode				S	Rn	Rd	rotate	immediate			
load/store immediate offset	cond	0	1	0	P	U	B	W	L	Rn	Rd	immediate				
load/store register offset	cond	0	1	1	P	U	B	W	L	Rn	Rd	shift amount		shift	0	Rm
load/store multiple	cond	1	0	0	P	U	S	W	L	Rn	register list					
branch/branch with link	cond	1	0	1	L	24-bit offset										

- S = For data processing instructions, updates condition codes
- S = For load/store multiple instructions, execution restricted to supervisor mode
- P, U, W = distinguish between different types of addressing_mode
- B = Unsigned byte (B==1) or word (B==0) access
- L = For load/store instructions, Load (L==1) or Store (L==0)
- L = For branch instructions, is return address stored in link register

X86 vs. ARM

- Fixed instruction formats of ARM
  - Simple decoding logic
  - Waste of memory space
  - Limited addressing modes

- Variable length formats of x86
  - Difficult to decode; sequential decoding
  - Compact machine codes
  - Accommodate versatile addressing modes

Types of ISA (CISC vs RISC)

- Complex instruction set computer (CISC)
  - Many instructions (several hundreds)
  - An instruction takes many cycles to execute
  - Example: Intel, AMD
- Reduced instruction set computer (RISC)
  - Small set of instructions (typically 32)
  - Simple instructions, each executes in one clock cycle – ***REALLY? Well, almost.***
  - Effective use of pipelining
  - Example: ARM, MIPS

Instruction Set Architecture (ISA)

NUMBER OF ADDRESSES

Number of Addresses

SUB Y,B ➔ 2-address instruction
SUB Y,A,B ➔ 3-address instruction

- Number of addresses per instructions is one way to describe processor architecture
- Number of addresses refers to *how many operand can an instruction take*.
  - The more the addresses – fewer number of instructions needed
  - The more the addresses – will require a longer instruction format
  - The more the addresses – the slower the fetch and execution
  - The more the addresses – will require a more complex processor
- With multiple-address instructions, there are commonly multiple general registers that can be used
  - Register references are faster than memory references
- Design trade-offs : choosing the number of addresses per instruction

Programs to Execute $Y = \frac{A - B}{C + (D \times E)}$

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

3 addresses → 4 instructions

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

2 addresses → 6 instructions

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

1 address → 8 instructions

(c) One-address instructions

0 address → 10 instructions

```
PUSH C
PUSH D
PUSH E
MUL
ADD
PUSH B
PUSH A
SUB
DIV
POP Y
```

The 3 Address Instruction

- 3 addresses
  - Operand 1, Operand 2, Result (Destination)
  - May be a forth address - next instruction (usually implicit, obtained from PC)
  - Example below: T=temporary location used to store intermediate results
  - Not common in use
  - Needs very long words to hold everything

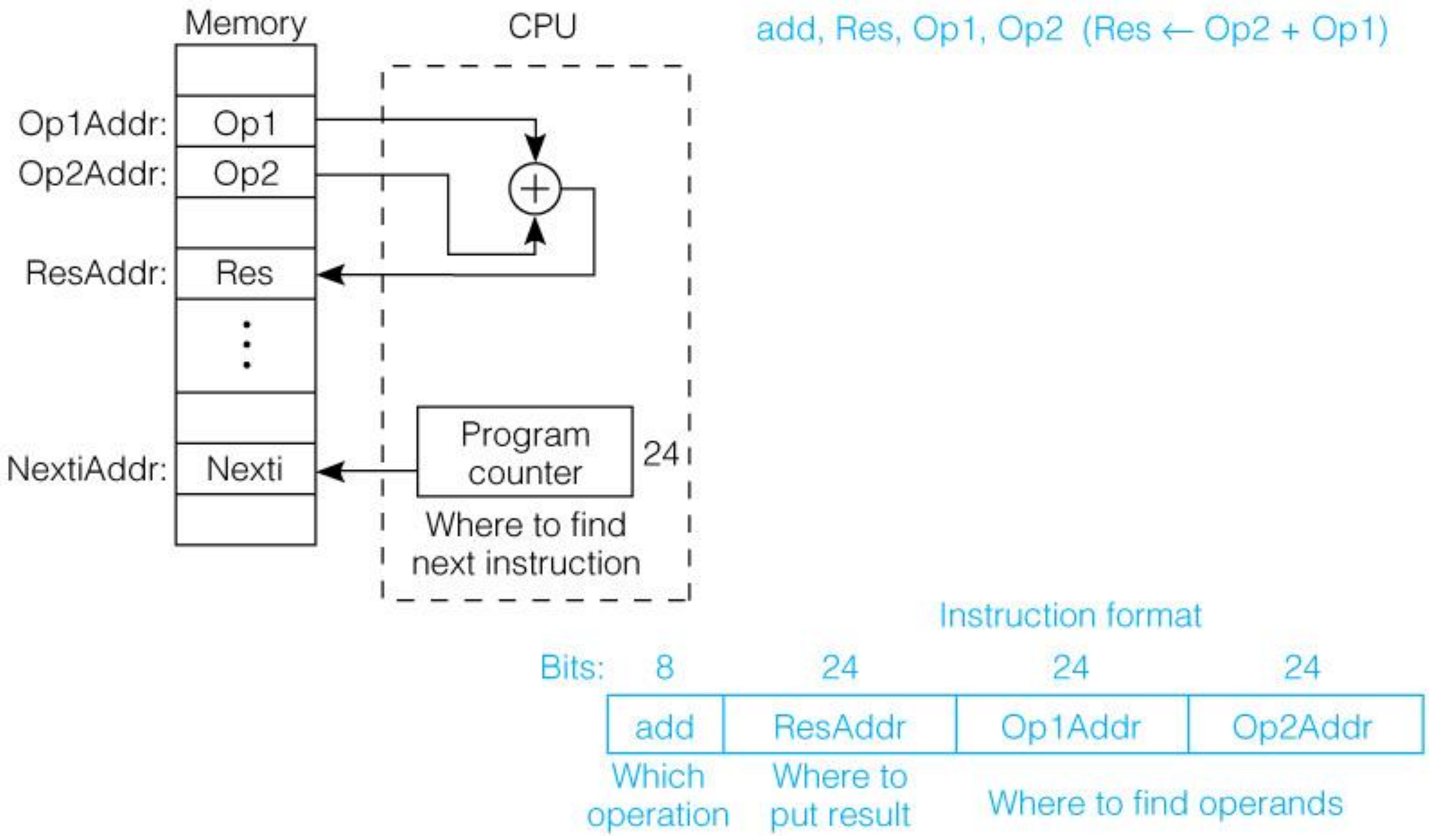
Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

3 addresses → 4 instructions

(a) Three-address instructions

The 3 Address Instruction



- Address of next instruction kept in a processor state register the - PC (Except for explicit Branches/Jumps)

The 2 Address Instruction

- 2 addresses
  - One address doubles as operand and result(destination)
  - Reduces length of instruction and space requirements
  - Requires some extra work
    - Temporary storage to hold some results
    - Done to avoid altering the operand value

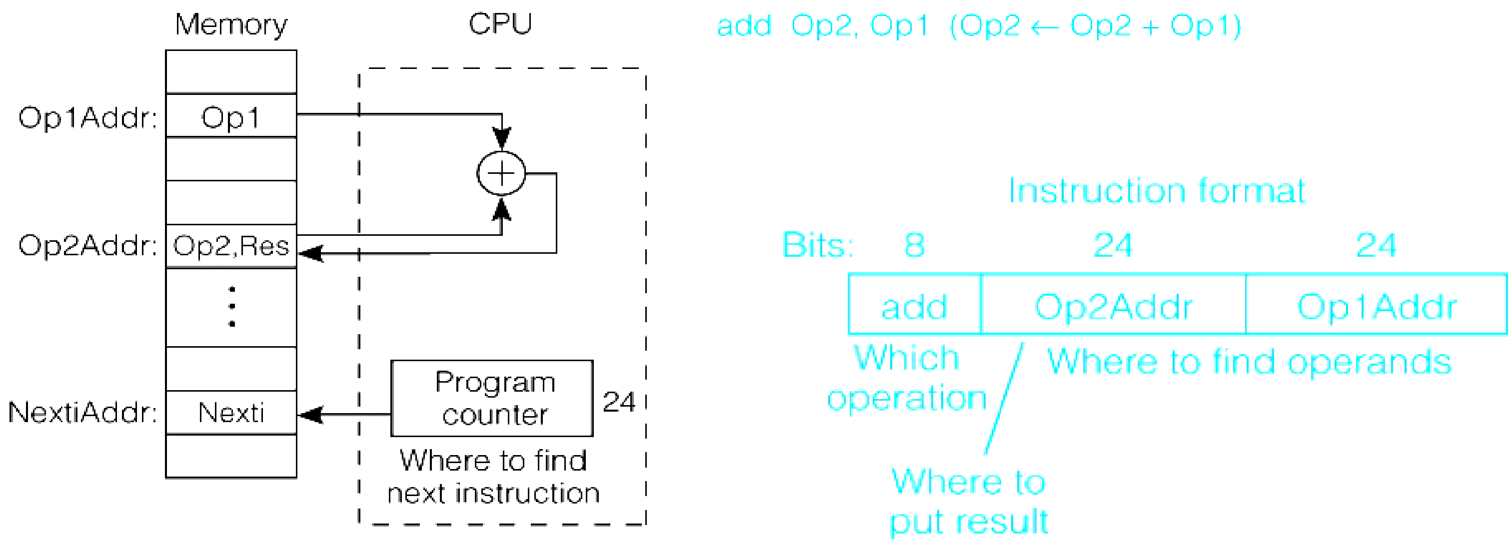
Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

2 addresses → 6 instructions

(b) Two-address instructions

The 2 Address Instruction



- Be aware of the difference between address, *Op1Addr*, and data stored at that address, *Op1*.
- Result overwrites operand 2, *Op2*, with result, *Res*
- This format needs only 2 addresses in the instruction but there is less choice in placing data

1 Address Instructions

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Programs to Execute $\mathbf{Y = \dfrac{A - B}{C + (D \times E)}}$

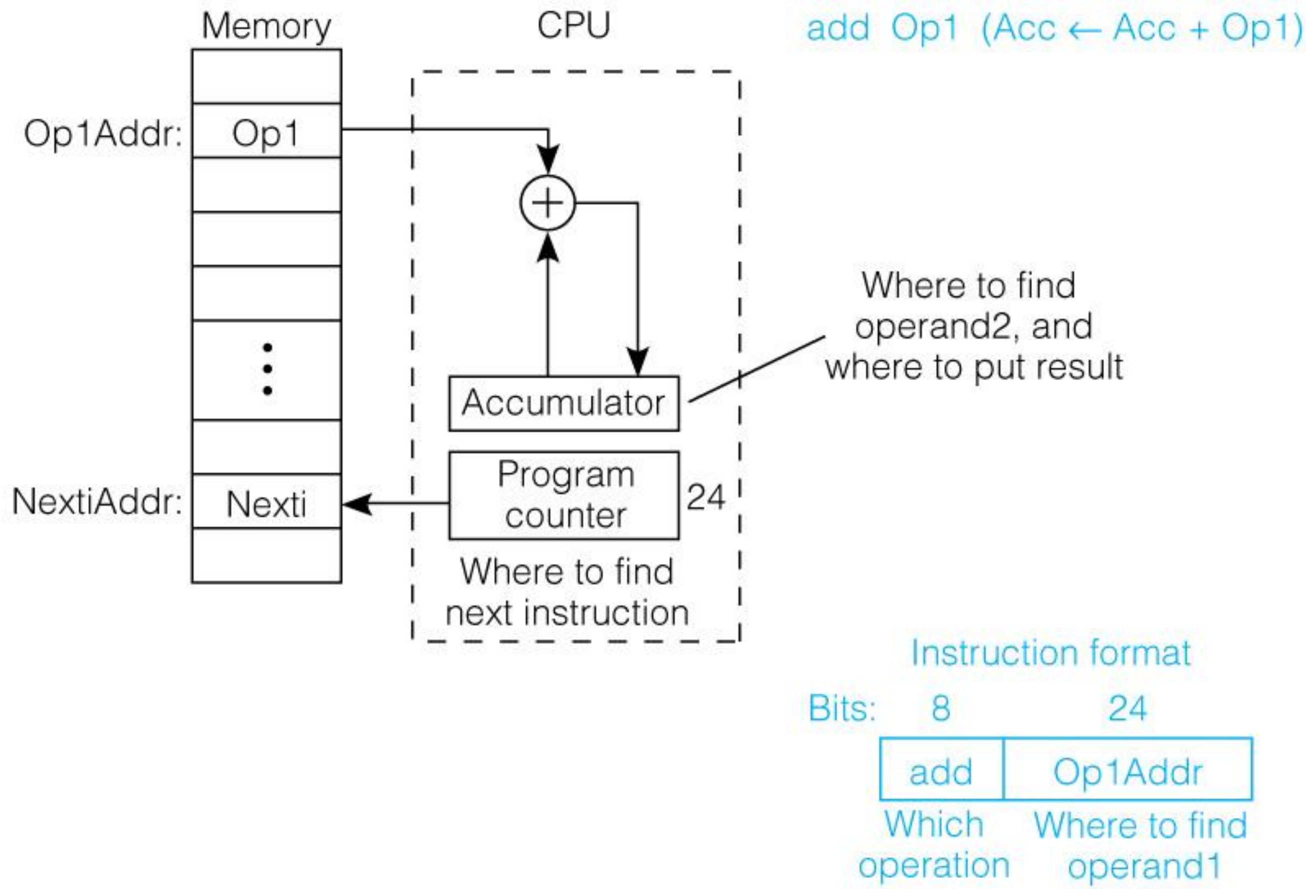
Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

1 address → 8 instructions

1 Address Instructions

We now need instructions to load and store operands:

LDA OpAddr
STA OpAddr



- Special CPU register, the accumulator, supplies 1 operand and stores result
- One memory address used for other operand

The 0 Address Instruction

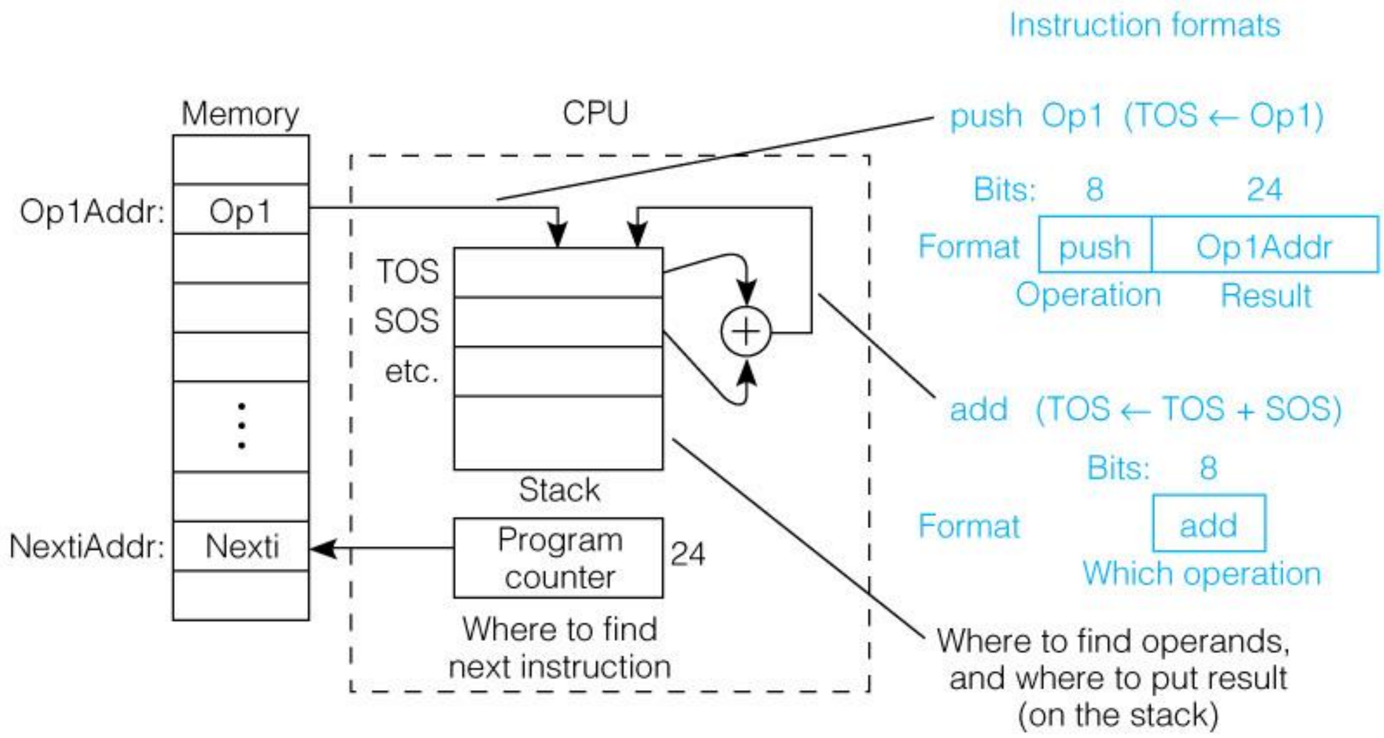
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - There are two Opcodes with one operand: PUSH *op*, POP *op*

Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

```
PUSH C
PUSH D
PUSH E
MUL
ADD
PUSH B
PUSH A
SUB
DIV
POP Y
```

0 address → 10 instructions

The 0 Address Instruction



- Uses a push down stack in CPU
- Arithmetic uses stack for both operands. The result replaces them on the TOS
- Computer must have a 1 address instruction to push and pop operands to and from the stack

Programs to Execute $Y = \frac{A - B}{C + (D \times E)}$

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

3 addresses → 4 instructions

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

2 addresses → 6 instructions

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

1 address → 8 instructions

(c) One-address instructions

0 address → 10 instructions

```
PUSH C
PUSH D
PUSH E
MUL
ADD
PUSH B
PUSH A
SUB
DIV
POP Y
```

How Many Addresses

- Number of addresses per instruction is a basic design decision
- More addresses
  - More complex (powerful?) instructions
  - Fewer instructions per program
  - More registers
    - Inter-register operations are quicker
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

… end of Part 2

- Hierarchy of Computer Languages
- General Concepts
- ISA Level
- Elements of Instructions
- **Instructions Types**
- **Instruction Formats**
- **Number of Addresses**
- Registers
- Types of Operands
- Addressing Modes

Part 2