COMPUTER ORGANIZATION ECE514

Chapter 2

Instruction Set Architecture

LEARNING OUTCOMES

- **Course Outcome (CO) - CO2**
- Describe the architecture and organization of computer systems

- **Program Outcome (PO) – PO1**
- Apply knowledge of mathematics, science and engineering fundamentals to the solution of complex electrical / electronic engineering problems
- L01-Knowledge in specific area-content

COURSE OUTLINE

- Machine instruction characteristics
- Types of operands and operations
- Addressing modes
- Assembly language & instruction formats

Learning Outcomes

- Describe the sequence of internal events as a computer executes an instruction.

MACHINE INSTRUCTIONS

- Machine language/codes that a computer can directly executed
- They provide the functional requirement for the processor (computer)
- They determine the operation of the processor
- The collection of different instructions that the processor can execute is referred to as the processor's instruction set

PROGRAM EXECUTION

- Basic function performed by a computer is execution of a program
- Program is a set of instructions stored in memory
- Memory consists of a set of "locations", defined by sequentially numbered addresses
- Each location may contain either instruction or data

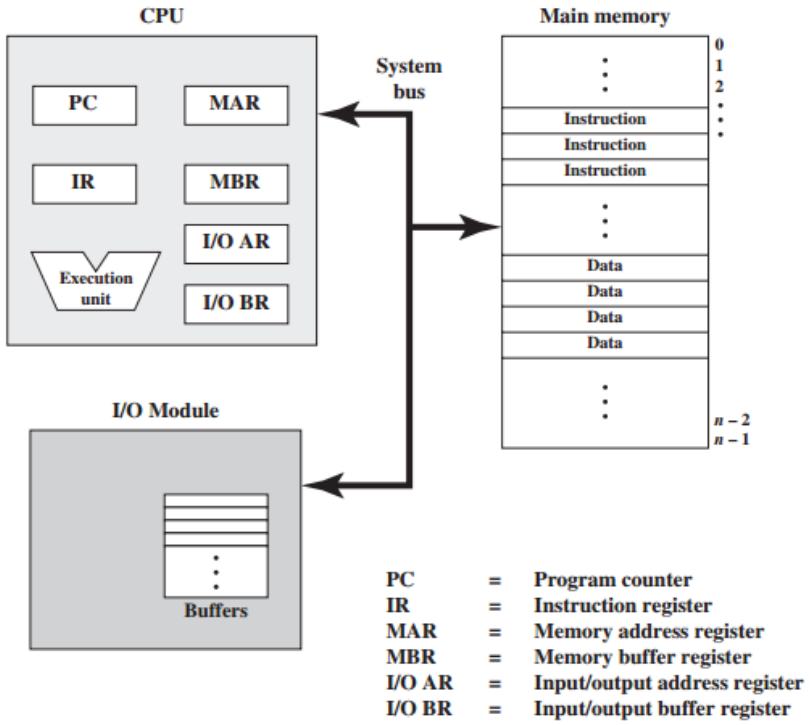


Figure 3.2 Computer Components: Top-Level View

PROGRAM EXECUTION

- Instruction processing consists of two steps: the CPU *fetches* (reads) instructions from memory one a time and *executes* each instruction
  - Fetch cycle and Execute cycle

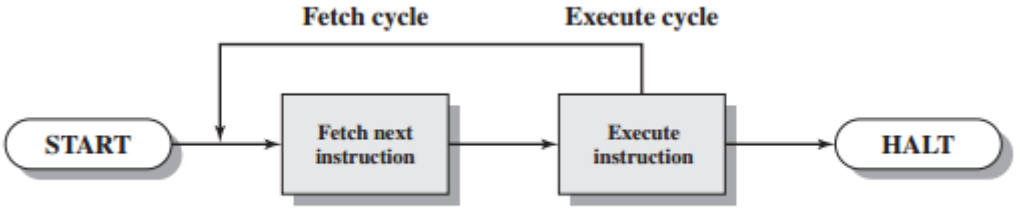


Figure 3.3 Basic Instruction Cycle

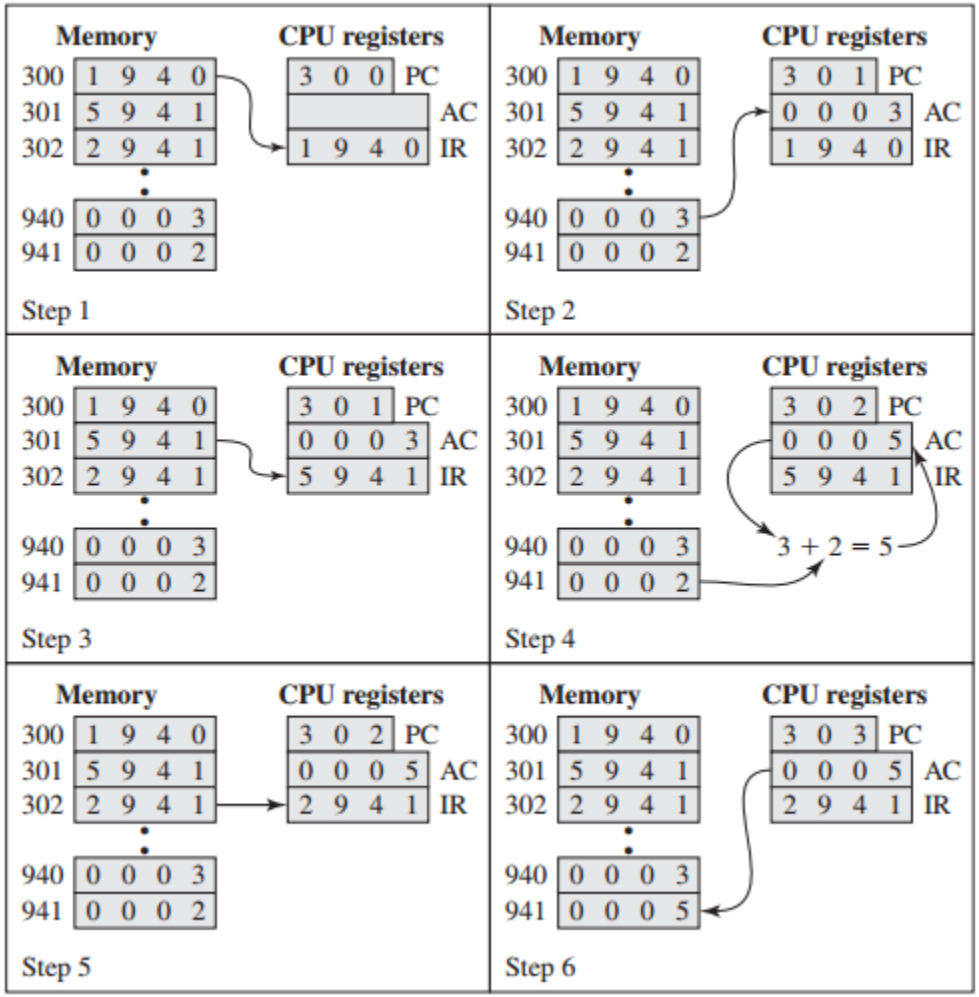
PROGRAM EXECUTION: INSTRUCTION CYCLE

- At the beginning of each instruction cycle, the processor fetches an instruction from memory
- A register called the program counter (PC) holds the address of the instruction to be fetched next
- Unless instructed otherwise, PC is always incremented after each instruction fetch, so that the processor will fetch the next instruction in sequence

PROGRAM EXECUTION: INSTRUCTION CYCLE

- E.g. PC is set to location 300, PC = 300.
- Processor fetches the instruction at location 300.
- On succeeding instruction cycles, processor will fetch instruction from locations 301, 302 and so on
- The fetched instruction is loaded into a register in the processor known as the instruction register (IR)

PROGRAM EXECUTION: INSTRUCTION CYCLE



MACHINE INSTRUCTION CHARACTERISTICS

- Each instruction must contain the information (elements) required by the processor for execution
- Elements of machine instruction:
  - Operation Code
  - Operand Reference
  - Next instruction reference

OPERATION CODE

- Specifies the operation to be performed by a binary code (opcode)
- All instructions must have opcode
- E.g. Addition, subtraction, store

OPERAND REFERENCE

- The operation may involve involve one or more operands (inputs for the operation)
- Source operand reference: Inputs for the operation
- Result operand reference: The operation may produce a result
- Operand reference can be in:
  1. Main or virtual memory
  2. Processor register
  3. Immediate
  4. I/O device

OPERAND REFERENCE

- Main memory - the reference is in the form of address must be supplied
- Processor register –
  - If one register available, the reference to it may be implicit (embed in instruction)
  - If more than one, then each register is assigned a unique name or number and the instruction must contain the desired register
- Immediate - the value of the operand is contained in the instruction
- I/O device – the instruction must specify the I/O module or it could be in the form of memory address (memory-mapped I/O)

NEXT INSTRUCTION REFERENCE

- Tells the processor where to fetch the next instruction after the execution of this instruction is complete
- In most cases, the next instruction to be fetched immediately follows the current instruction
  - Thus no explicit reference to the next instruction
- When explicit reference is needed
  - The reference could be main memory or virtual memory

INSTRUCTION REPRESENTATION

- Instructions are sequence of bits
- Example of instruction representation
- 2^4 = 16 opcodes
- 2^6 = 64 operands
- Two operands

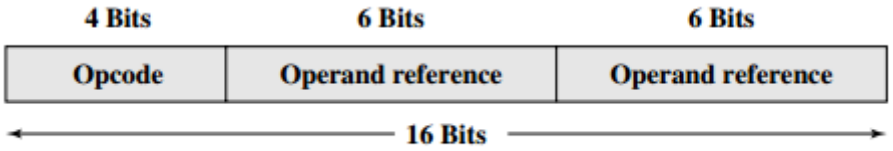


Figure 10.2 A Simple Instruction Format

INSTRUCTION REPRESENTATION

Opcode

- Represented by symbolic representation (mnemonics) that indicate the operation.
- e.g. ADD for Addition & SUB for Subtraction

INSTRUCTION REPRESENTATION

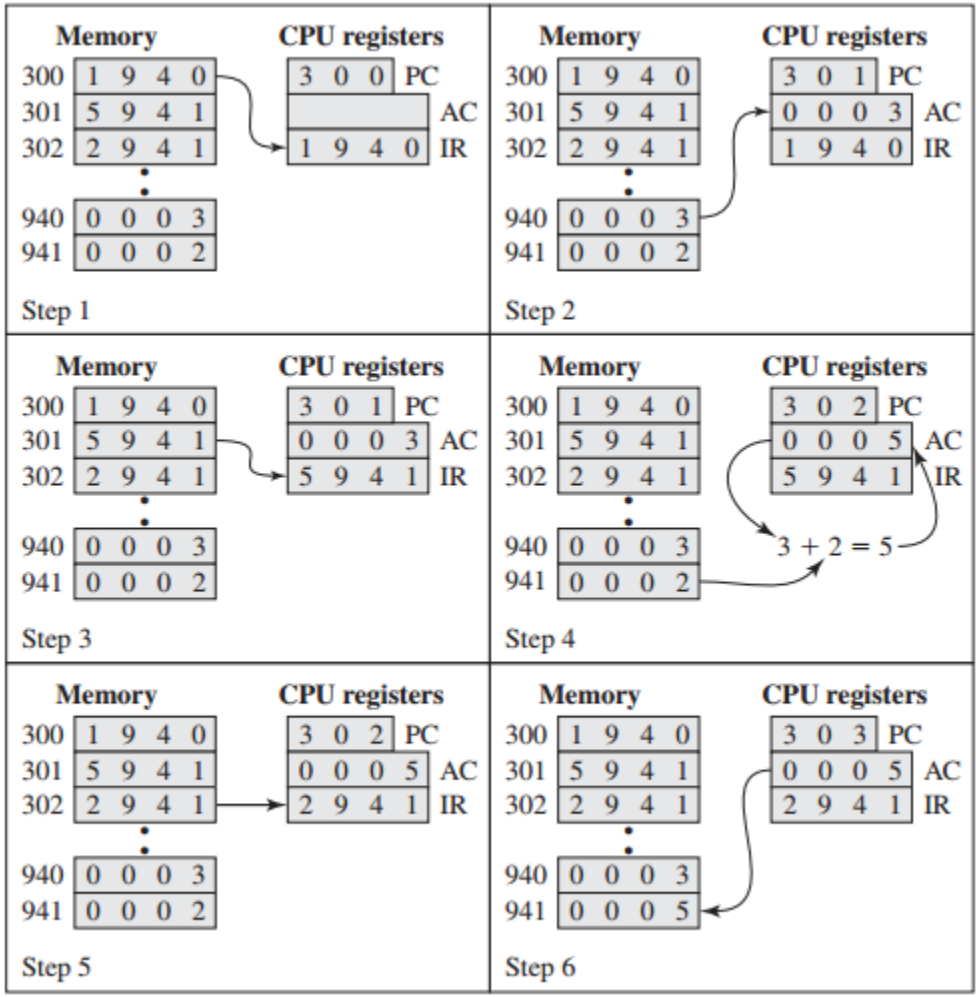
Opcode

- Represented by symbolic representation (mnemonics) that indicate the operation.
- e.g. ADD for Addition & SUB for Subtraction

Operands

- Represented by symbolic representation.
- e.g. ADD R, Y - Mean add the value contain in data location Y (address in memory) to the contents of register R (in processor).

IDENTIFY OPCODE & OPERANDS



INSTRUCTION TYPES

- In high-level language, expression X = X + Y can be easily expressed
- X = memory location 513 and Y = memory location 514
- In low-level language (machine language) context,
  1. Load a register with the contents of memory location 513
  2. Add the contents of memory location 514 to the register
  3. Store the contents of the register in memory location 513

INSTRUCTION TYPES

- Data processing: Arithmetic and logic instructions
- Data storage: Movement of data into or out of register and or memory locations
- Data movement: I/O instructions
- Control: Test and branch instruction

NUMBER OF ADDRESSES

- Number of address involve in operation
- What is the maximum number of addresses one might need in an instruction?
  - One source operand?
  - Two source operands?
- The result of an operation must be stored
  - Third address to reference destination operand

- A program to execute Y = (A - B) / (C + (D * E))

THREE-ADDRESS INSTRUCTIONS

- Two source operand locations and one destination operand location
- Temporary location 'T' is used to store some intermediate results so as not to alter the value in any operand location
- Not common and rarely used because require long instruction format to hold three address references

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	$Y \leftarrow Y \div T$

TWO-ADDRESS INSTRUCTIONS

- One address must do double duty as both operand & result
- 'MOVE' instruction is used to move one of the values to a result or temporary location before performing the operation
- Reduces length of instruction thus the space requirement, very common in instruction sets

Instruction	Comment
MOVE Y, A	Y ← A
SUB Y, B	Y ← Y − B
MOVE T, D	T ← D
MPY T, E	T ← T × E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y ÷ T

ONE-ADDRESS INSTRUCTIONS

- Second address must be implicit
- Use an accumulator, 'AC' (processor register) which implied the second address
- AC contains one of the operands and is used to store the result
- Simpler than the above
- Common in early machines

Instruction	Comment
LOAD D	AC ← D
MPY E	AC ← AC × E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC − B
DIV Y	AC ← AC ÷ Y
STOR Y	Y ← AC

ZERO-ADDRESS INSTRUCTIONS

- All addresses are implicit, as in register based operations
- Uses a stack (special memory organization: last-in-first-out set of locations)
- The stack is in a known location
- (Often) top two elements are in processor registers
- The instructions would reference the top two stack elements

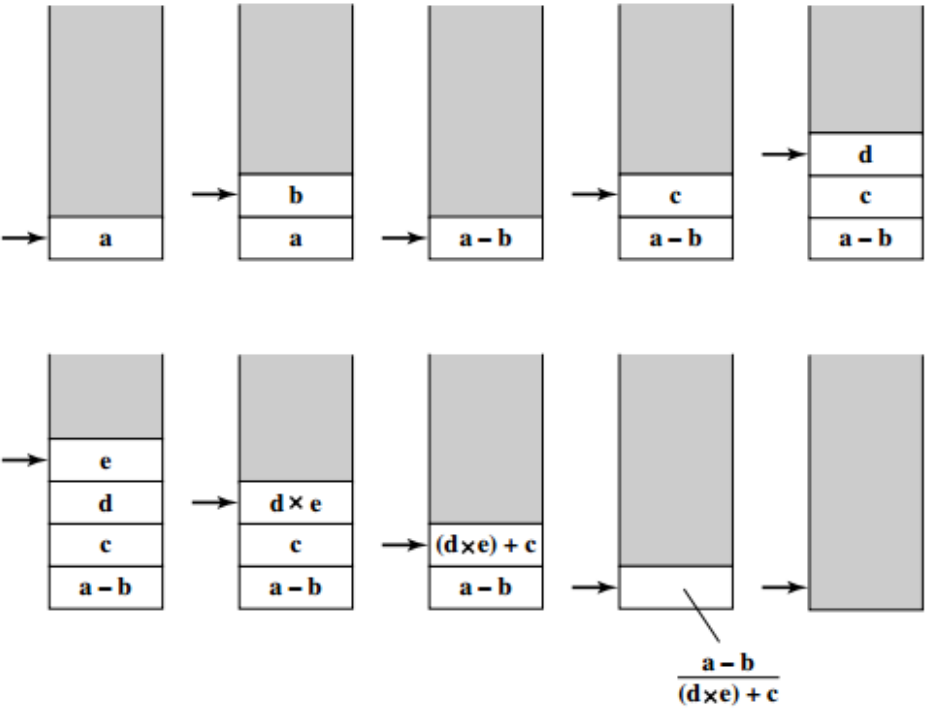


Figure 10.16 Use of Stack to Compute f = (a − b)/[(d × e) + c]

	Stack	General Registers	Single Register
	Push a Push b Subtract Push c Push d Push e Multiply Add Divide Pop f	Load R1, a Subtract R1, b Load R2, d Multiply R2, e Add R2, c Divide R1, R2 Store R1, f	Load d Multiply e Add c Store f Load a Subtract b Divide f Store f
Number of instructions	10	7	8
Memory access	10 op + 6 d	7 op + 6 d	8 op + 8 d

Figure 10.15 Comparison of Three Programs to Calculate $f = \frac{a - b}{c + (d \times e)}$

MORE ADDRESS

- (Typically) more general purpose registers
- Faster execution of program (inter-register operations are quicker)
- Fewer instructions per program

- More complex instructions
- More complex processor

LESS ADDRESS

- (Shorter) More primitive instructions / Less complex instructions
- Less complex processor

- More instructions per program
- Longer, more complex programs
- Longer execution time

INSTRUCTION SET DESIGN ISSUES

- Operation repertoire
  - How many and which operations to provide, and how complex operations should be.
- Data Types
  - The various types of data upon which operations are performed.
- Instruction Formats
  - Instruction length (in bits), number of addresses, size of various fields, and so on.
- Registers
  - Number of processor registers that can be referenced by instructions and their use
- Addressing
  - The mode or modes by which the address of an operand is specified

ADDRESSING MODES AND FORMATS

ADDRESSING MODES

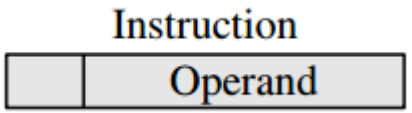
Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Table of Basic Addressing Mode

* A = contents of an address field in the instruction
* R = contents of an address field in the instruction that refers to a register
* EA = actual (effective) address of the location containing the referenced operand
*(X) = contents of memory location X or register Y

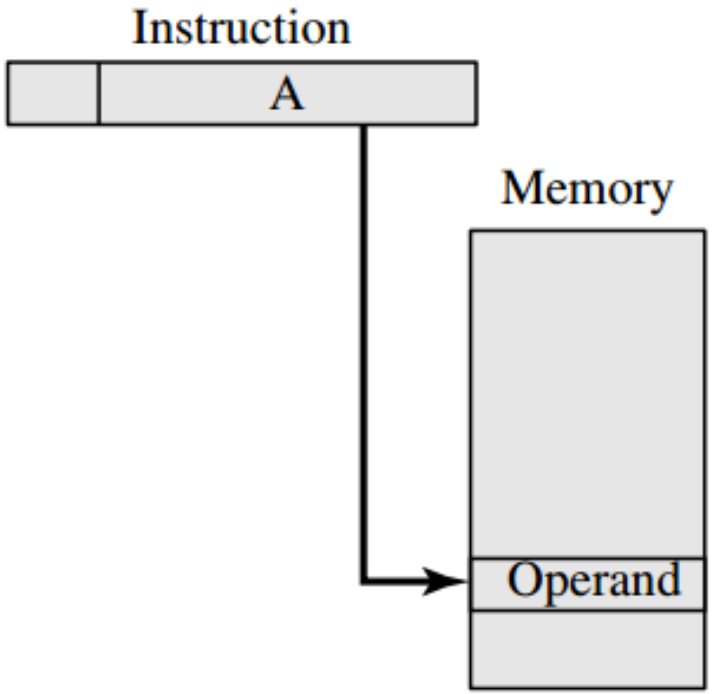
IMMEDIATE ADDRESSING

- Operand value is part of instruction
- Operand = address field
- e.g. ADD 5
- Add 5 to contents of accumulator
- 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



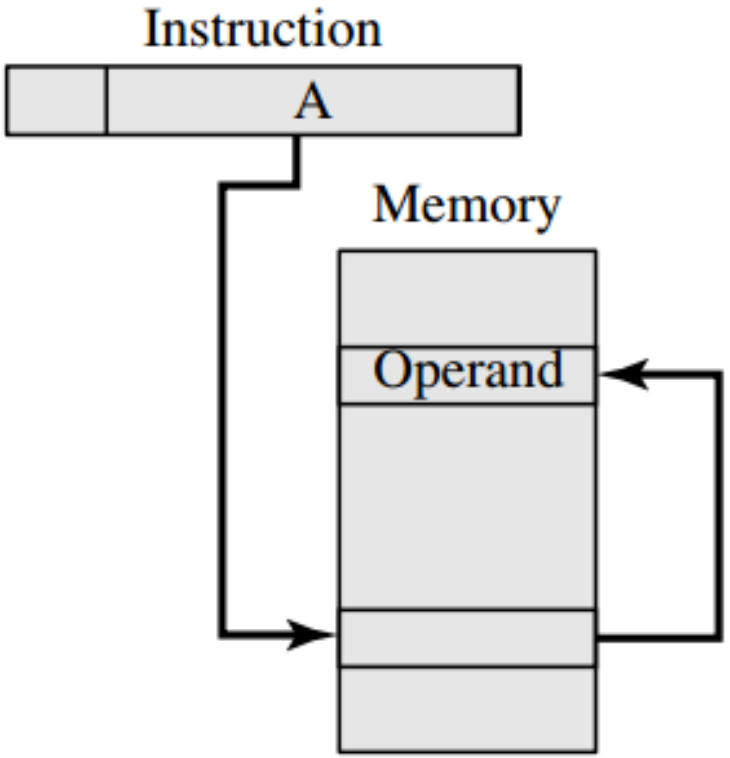
DIRECT ADDRESSING

- Address field of instruction contains address of operand in memory
- 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



INDIRECT ADDRESSING

- Memory location pointed to by the address field contains the address of memory where the operand is located.
- 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

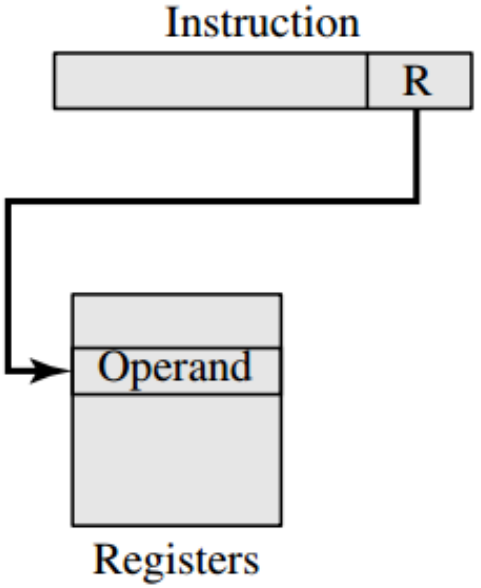


INDIRECT ADDRESSING

- Large address space
  - 2^n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A))) * Draw the diagram yourself
- Multiple memory accesses to find operand
  - Hence slower

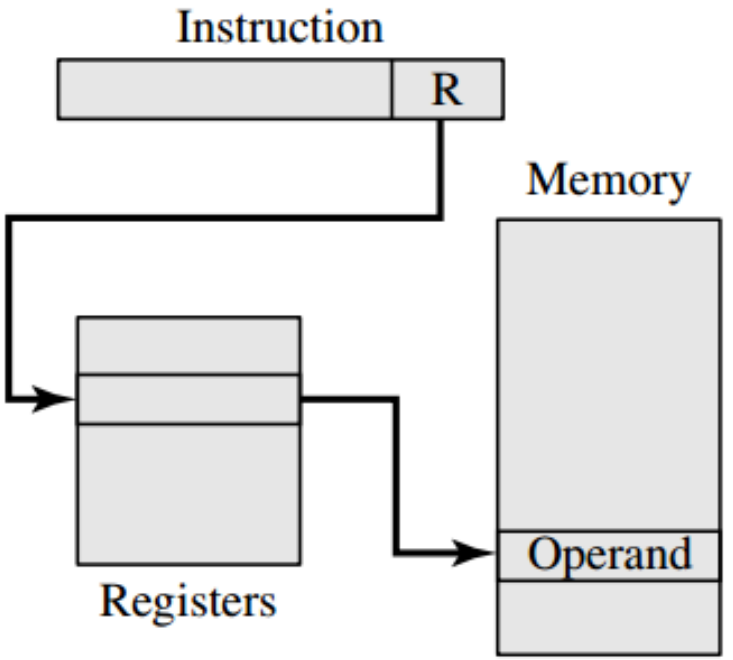
REGISTER ADDRESSING

- The operand is held in a register named in the address field of instruction
- EA = R
- Very small address field needed
  - 2^n where n is the number of registers
- Shorter instructions and faster instruction fetch
- No memory access hence very fast execution
- Very limited address space
- Implies processor registers are heavily used if register addressing is used
  - Decide which values remain in registers and which should be stored in memory



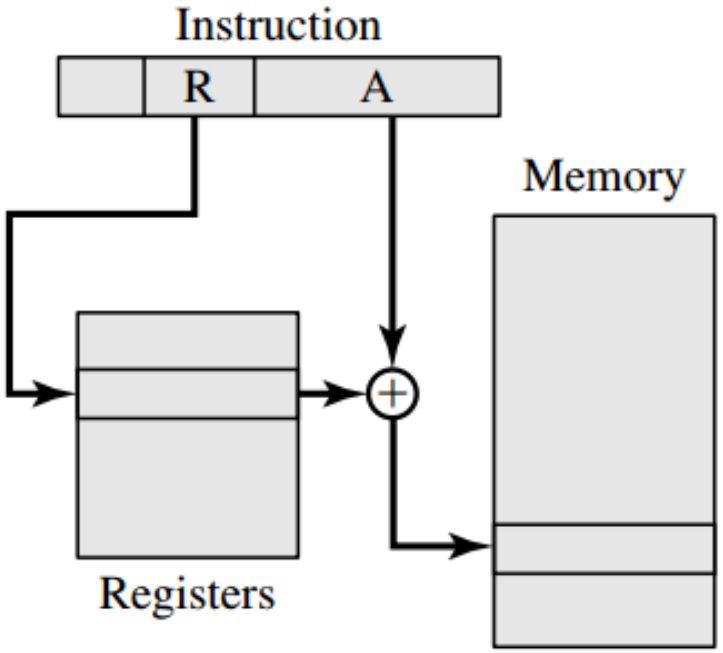
REGISTER INDIRECT ADDRESSING

- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing



DISPLACEMENT ADDRESSING

- Combines direct addressing and register indirect addressing
- EA = A + (R)
- Address field hold two values
- * A = base value
- * R = register that holds displacement
- * or vice versa
- Relative
- Base-register
- Indexing



RELATIVE ADDRESSING

- PC-relative addressing
- R = PC (Program counter)
- EA = A + (PC)
- The next instruction address is added to the address field (A)

BASE-REGISTER ADDRESSING

- Address (A) field holds the displacement
- Register (R) holds the bass address
- R may be explicit or implicit (embed in instruction)
- EA = A + R

INDEXED ADDRESSING

- Address (A) field holds the base address
- Register (R) holds the displacement – index register
- R may be explicit or implicit (embed in instruction)
- EA = A + R
- Mechanism for performing iterative operations
  - e.g accessing arrays
- * EA = A + R
- * R++

INDEXED ADDRESSING

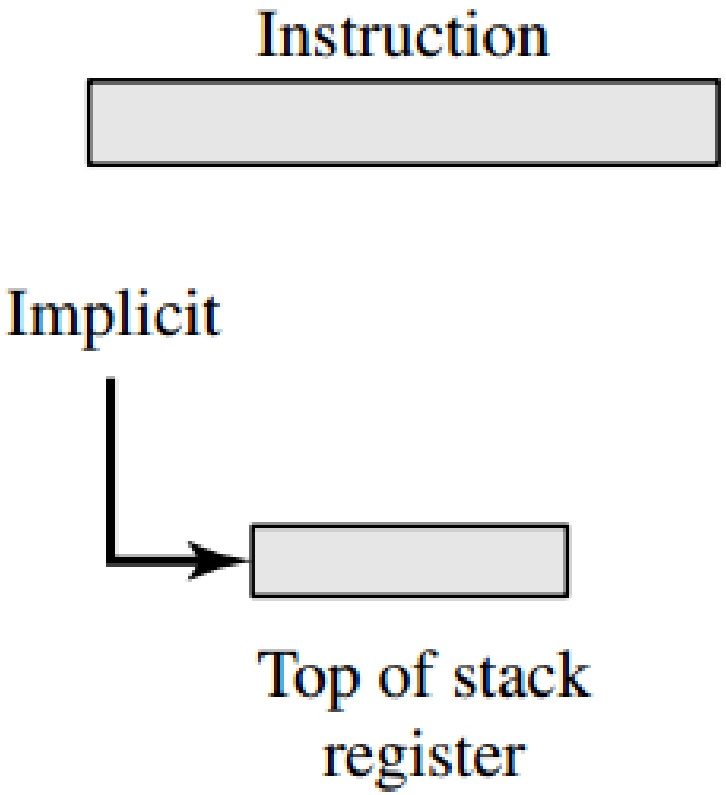
- Iterative task is a common in CPU processing
  - Increment and decrement the index register after each reference
- Some systems with devoted index register can automatically increment/decrement as part of the same instruction cycle (i.e. after reference)
  - Autoindexing
  - Special instruction for increment/decrement

INDEXED ADDRESSING

- Postindex – indexing is performed after the indirection
- EA = (A) + (R)
- Preindex – indexing is performed before the indirection
- EA = (A+(R))

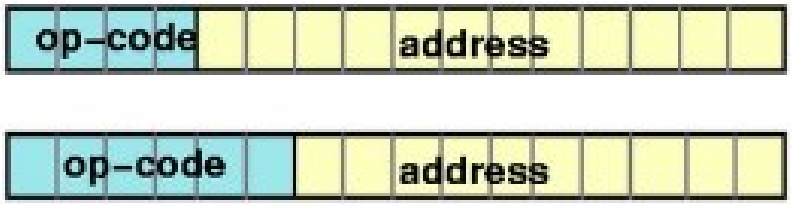
STACK ADDRESSING

- Operand is implicitly referenced – embed in instruction
- A register is devoted for stack (Stack Pointer – SP)
- Points to the top of the stack
- psha, pulb



EXPANDING OPCODES

- For a given instruction format length, there is a trade-off between the number of bits used for the opcode and the number of bits used for operands (e.g., addresses)
- More opcode bits means more operations
- More address bits means more addressable locations
- Short opcodes → Short instructions

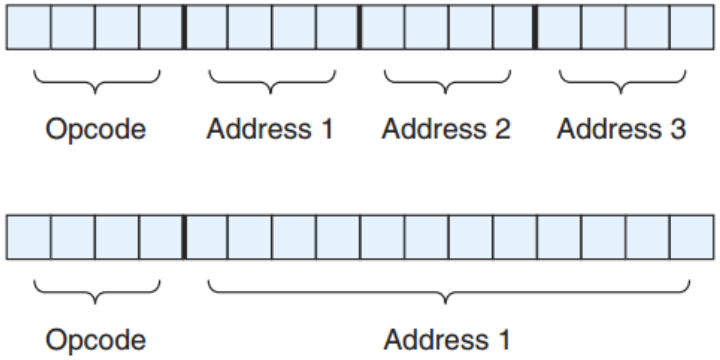


EXPANDING OPCODES

- Some opcodes short and have a means to provide longer ones when needed
- Short opcode, a lot of bits are left to hold operands (two or more operands per instruction)
- Instructions that do not need operands, all the bits can be used for the opcode e.g. Halt
- By using specific prefixes of the opcode, you can get varying number of variable length of opcodes

EXPANDING OPCODES

- Consider a machine with 16-bit instructions and 16 registers

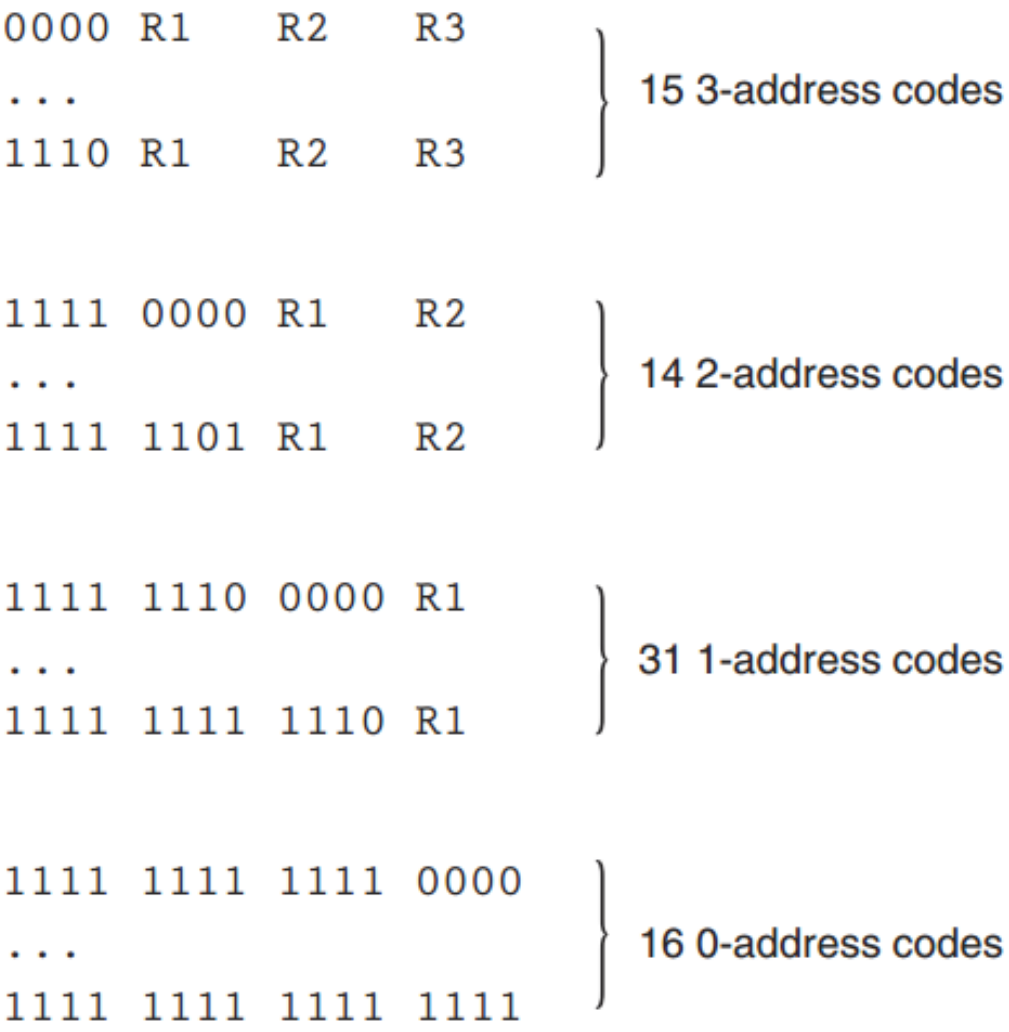


EXPANDING OPCODES

- Suppose we wish to encode the following instructions
- 15 instructions with 3 addresses
- 14 instructions with 2 addresses
- 31 instructions with 1 addresses
- 16 instruction with 0 addresses

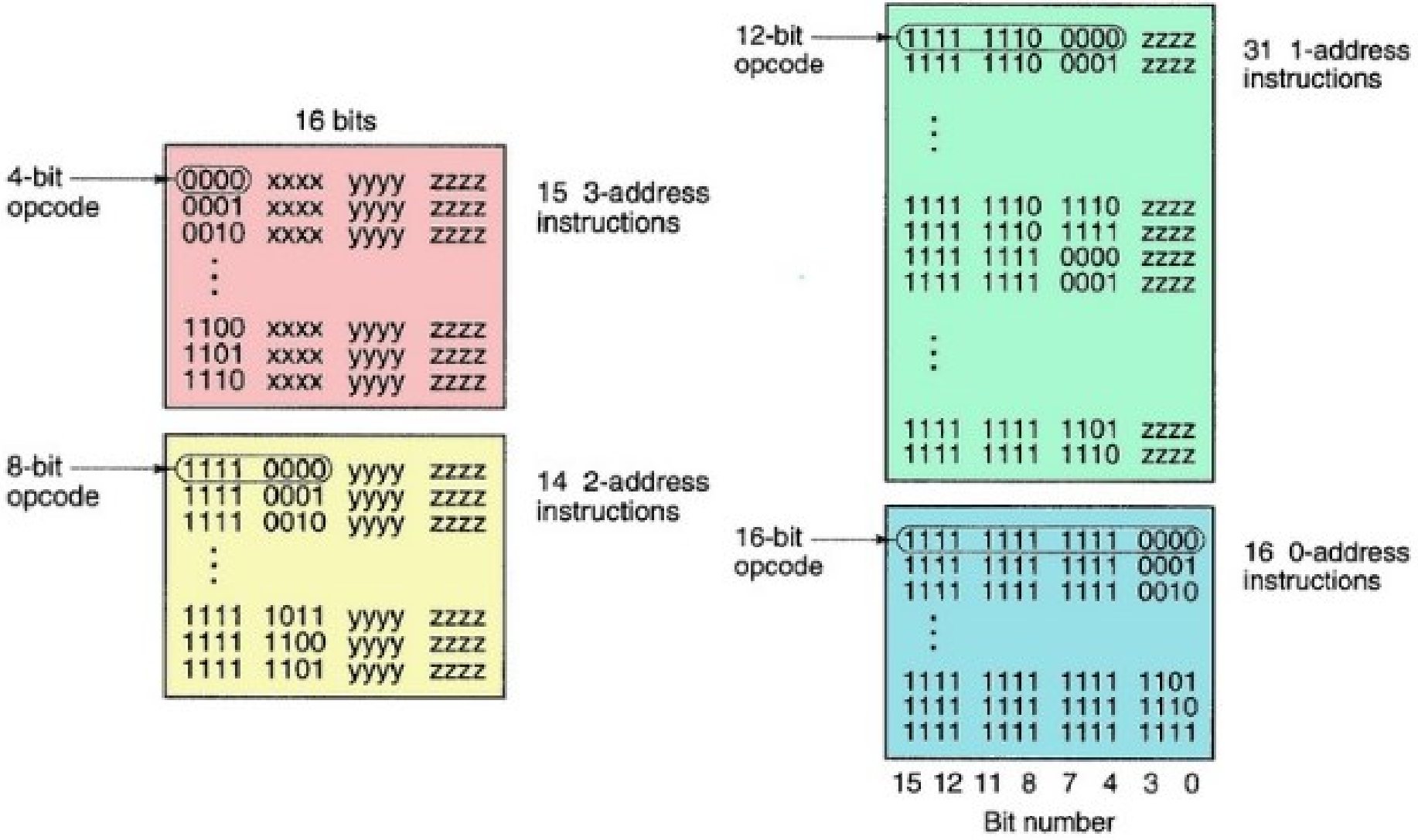
- Can we encode this instruction set in 16 bits?

EXPANDING OPCODES



- This expanding opcode scheme makes the decoding more complex
- Instead of simply looking at a bit pattern and deciding which instruction it is

EXPANDING OPCODES



EXPANDING OPCODES

- To decode the instructions

```
if (leftmost four bits != 1111 ) {
   Execute appropriate three-address instruction}
else if (leftmost seven bits != 1111 111 ) {
   Execute appropriate two-address instruction}
else if (leftmost twelve bits != 1111 1111 1111 ) {
   Execute appropriate one-address instruction }
else {
   Execute appropriate zero-address instruction
}
```

EXERCISE 1

- Construct a program using one-address instructions that will solve for Y on given equation. Remember that one-address instructions always use the accumulator (Acc) in the instructions.

$$Y = \frac{(A + B)}{(C - D * E)}$$

EXERCISE 2

- Compute the equation R = A * C * C - D / B using the following instructions. Assume the

  i) two-address

  ii) one-address

EXERCISE 3

- Produce the complete instruction codes for a certain processor that has:
- An instruction length of 12 bits, with the size of its address field 4 bits.
- The instruction set should have 15 two-address instructions, 14 one-address instructions and the rest as zero-address instructions.
- Use expanding op-code technique.

EXERCISE 4

- Basic computer system has 60 instructions in its instruction set and 3 bits of address field. Each instruction is made up of 12 bits. The instruction set is to be designed so that it has 5 three-address instructions, 15 two-address instructions, 30 one-address instructions and 10 zero-address instructions. Answer the following questions:

  i) determine the maximum number of bits required for the address field,

  ii) determine the minimum number of bits required for the op-code field and

  iii) produce complete instruction codes for the system using expanding op-code

  technique.

EXERCISE 5

- A certain computer has 200 instructions in its instruction set. All of the instructions are one-address instructions and each instruction is made up of 24 bits. Determine :-

i. The number of bits required to make the operation code (op-code) part.

ii. The number of bits allocated for the address part.

iii. The maximum size of memory the computer can have.

iv. The address range of the memory in hexadecimal.

v. The size of the data and address bus.