مدينة زويل للعلوم والتكنولوجيا
Zewail City of Science and Technology

COMMUNICATION AND INFORMATION ENGINEERING

CIE 314
Embedded Systems Fundamentals

Lecture #5
CPU Instruction Set

Instructor:

Dr. Ahmad El-Banna

SPRING 2017

Agenda

- Register Transfer Notation
- Machine Language and Assembly Instructions
- Types of Instructions
- The Stack and the Stack Pointer
- Addressing Modes
- Orthogonal CPU's
- Design Tips

3.7 CPU INSTRUCTION SET

- Register Transfer Notation
- Machine Language and Assembly Instructions
- Types of Instructions
- The Stack and the Stack Pointer
- Addressing Modes
- Orthogonal CPU's

CPU Instruction Set

- The architecture and operation of a CPU are intimately related to:
  - how its instructions are organized in the Instruction Set and
  - how instructions access data operands, the Addressing Modes.
- Instructions for the CPU are stored in memory just as any other word.
- What makes it an instruction is the fact that it is going to be decoded by the CPU during the instruction cycle, to find out what to do.
- let us first introduce useful notation to describe instructions.

REGISTER TRANSFER NOTATION (RTN)

- RTN: a notation for MCU operations
  - Independent from specific MCU architecture
  - Takes in consideration the system features
- Source/destination Notation

  destination ← *source*
- Programmer's Model Operands
  - Constants are expressed by their value

    Ex.: 24, 0xF230, MyConstant
  - Registers are referred by their names

    Ex.: Rn (Ex.: R5, R3, R7)
  - Memory and I/O locations are referred as (address) or (Rn)

    Ex.: (0x345A) = "*the datum a memory address 345Ah*"

    (R5) = "*the datum a memory address given by R5*"

EXAMPLE OF RTN OPERATIONS

Example 3.10 *Assume two 16-bit registers,* R6 *and* R7, *with contents* R6 = 4AB2h *and* R7 = 354Fh, *respectively. Moreover, assume words at addresses* [4AB2h] = 02ACh, [4C26h] = 94DFh *and* [4AB8h] = 3F2Ch. *Assume little endian convention when necessary. The following examples illustrate more RTN expressions:*

RTN	Meaning	Result
R6 ← 3FA0h	Load R6 with 3FA0h	R6 = 3FA0h
R7 ← R6	Copy in R7 the current contents of R6	R7 = 4AB2h
R7 ← (R6)	Copy in R7 the word whose memory address is stored in R6. Also: Copy in R7 the word in memory pointed at by R6.	R7 = 02ACh
byte(4C26h) ← byte(4C26h) − (R6)	Subtract from the byte at address 4C26h the byte whose address is given by R6	[4C26h] = 9433h
(R6) ← (R6) + R7	Add the contents of R7 to the word pointed at by R6.	[4AB2h] = 37FBh
R7 ← (R6 + 6h)	Copy in R7 the memory word whose address is given by R6 + 6h	R7 = 3F2Ch
byte(R6) ← (R6) − 0x92	subtract 0x92 from the current memory byte at address given by contents of R6.	[4AB2h] = 021Ah

MACHINE VS. ASSEMBLY LANGUAGE

- Machine Language Instructions
  - Sequence of zeros and ones understood by the CPU
  - Hard to read by humans
  - Consists of several fields
    - Opcode, source and destination fields, and an optional datum
- Assembly Language Instructions
  - A human understandable notation for machine language
  - One assembly instruction per machine language instruction
  - Consists of several fields
    - A mnemonic followed by zero or more operands
- Assembly Process
  - Converts an assembly language program into a machine language program

MACHINE CODE AND ASSEMBLY PROCESS (1/2)

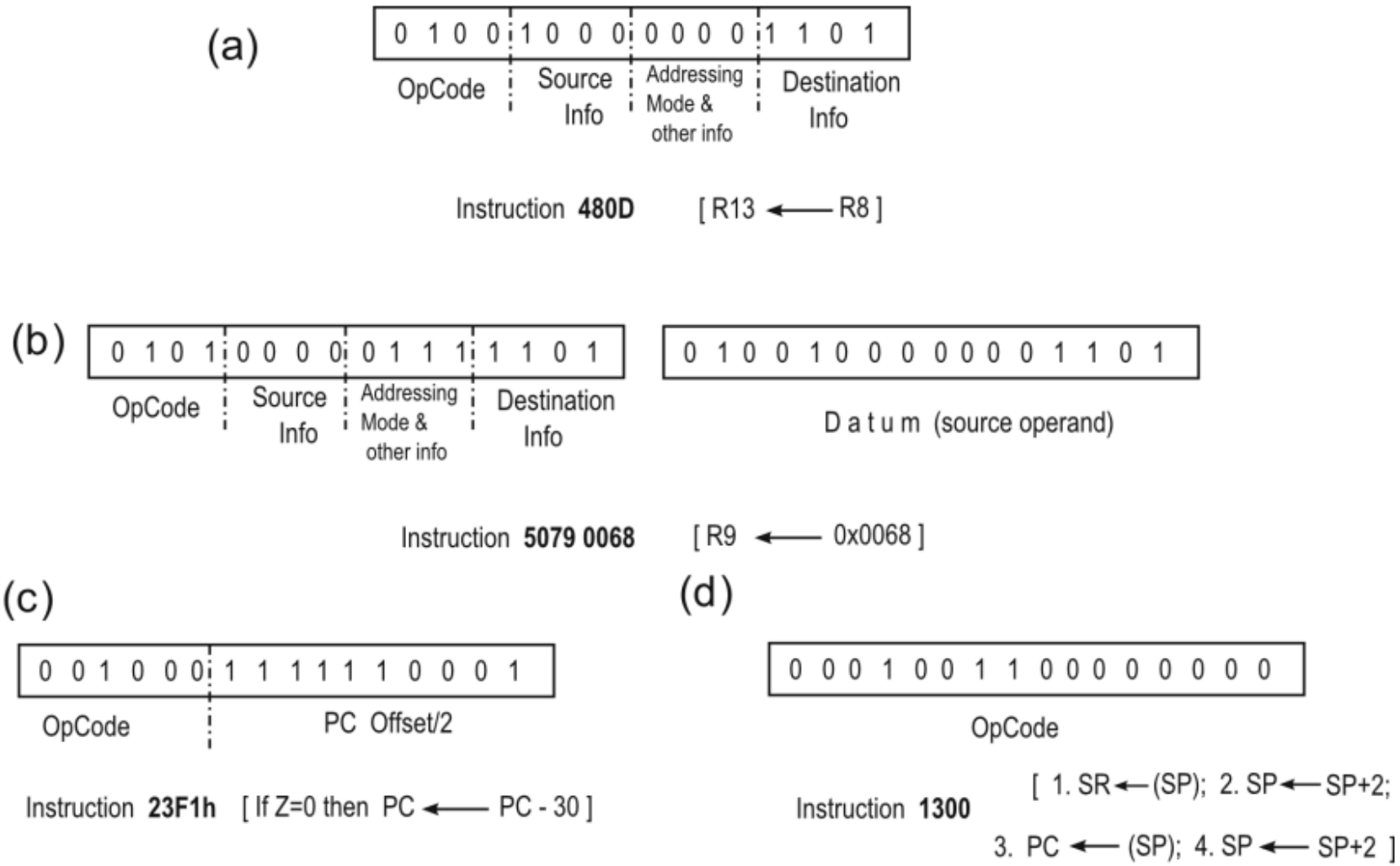


Fig. 3.23 Four machine language instructions for the MSP430

MACHINE CODE AND ASSEMBLY PROCESS (2/2)

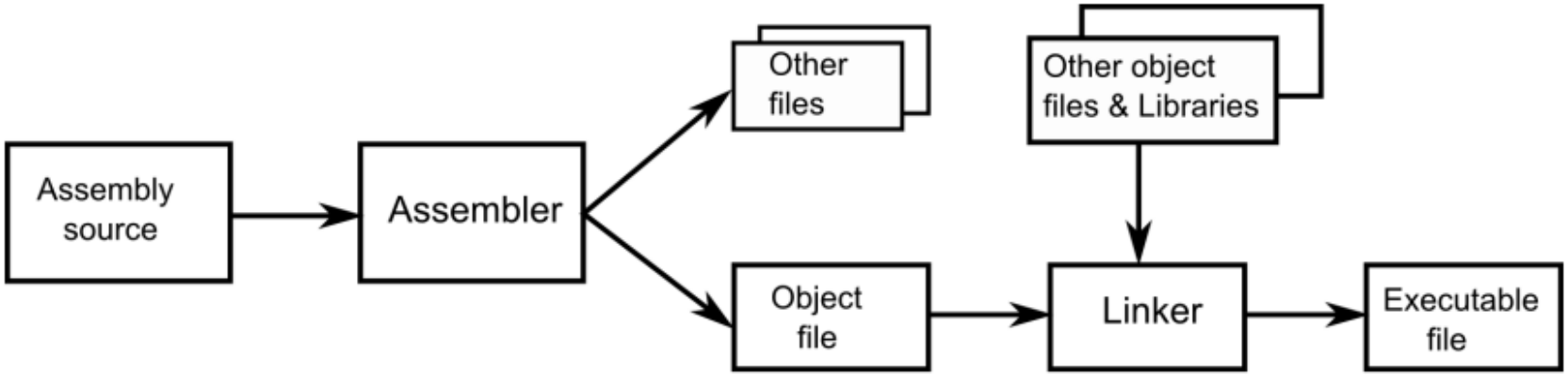


Fig. 3.24 Basic assembly process

Machine language	Assembly language
480D	`mov R8,R13`
5079 006B	`add.b #0x6B,R9`
23F1	`jnz 0x3E2`
1300	`reti`

Assembly encoding for MSP430 machine instructions

INSTRUCTION TYPES

- Data Transfer Instructions
  - Copy data from a source to a destination
- Arithmetic-logic Instructions
  - Perform arithmetic and/or logic operations on operands
- Program Control Instructions
  - Modify the default flow of execution in a program

DATA TRANSFER INSTRUCTIONS

- Copy data from a source to a destination

  destination ← *source*

- Do not affect flags
- Included Instructions:
  - Data transfer: MOVE
  - Data exchange: SWAP
  - Stack manipulation: PUSH & POP
- Treat I/O locations like memory
  - Memory-mapped I/O
- Examples:

```
MOV R8,R3 ; Copies the contents of R8 into R3
MOV (0xF348),R5 ; Copies into R5 the word at address F348h
PUSH R7 ; Copies onto the top of the stack the contents of R7
```

ARITHMETIC-LOGIC INSTRUCTIONS

- Perform arithmetic and/or logic operations on data

  destination ← (*DestinationOperand* ∅ *SourceOperand*)

- Flags affected according to operation result
- Included Instructions:
  - Arithmetic: ADD, SUB
  - Compare and test: CMP, TEST
  - Bitwise logic: AND, OR, XOR, NOT
  - Bit Displacement: SHIFT, ROTATE
- Examples:

```
ADD R7,R5 ; Places on R5 the sum of the contents of R5 and R7
AND #05AD,R6 ; Places on R6 the bitwise result of anding the
               contents of R6 and the value 05ADh
ROTL R3 ; Rotates the contents of register R3 one position to the
          left
```

WORKING WITH BITS

- **Bitwise operations work directly on bits**

Table 3.4 Logic properties and applications

`0.AND.X = 0;`	To **clear** specific bits in destination, the binary expression of the source has 0 at the bit positions to clear and 1 elsewhere (Fig. 3.25a)
`1.AND.X = X`	
`0.OR.X = X;`	To **set** specific bits in destination, the binary expression of the source has 1 at the bit positions to set and 0 elsewhere (Fig. 3.25b)
`1.OR.X = 1`	
`0.XOR.X = X;`	To **toggle or invert** specific bits in destination, the binary expression of the source has 1 at the bit positions to invert and 0 elsewhere (Fig. 3.25c)
`1.XOR.X =` $\overline{X}$	

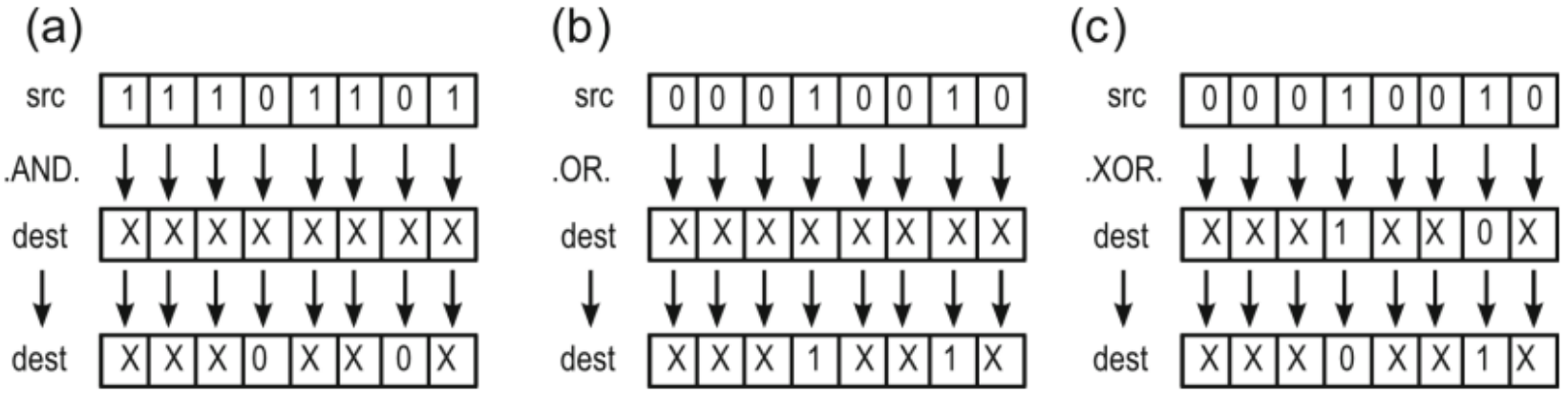


Fig. 3.25 Using logic properties to work with bits 1 and 5 only

PROGRAM CONTROL INSTRUCTIONS

- Modify the default flow of execution in a program

  $PC \leftarrow NewAddress$

- Do not affect flags
- Included Instructions:
  - Unconditional Jump: Always change the PC
  - Conditional Jump: Change the PC if condition is true
  - Subroutine Calls and Returns: Transfer control from main to subroutines, returning to the calling point
- Examples:

```
JMP #F345h ; Loads PC with the address 0xF345 so program execution
           continues there
JZ #F345 ; Loads PC with the address 0xF345 if the Zero Flag is set
CALL Sub1 ; Saves PC onto the stack and loads PC with address Sub1.
           When special instruction RET is executed, the PC is
            restored from the stack
```

CONDITIONAL JUMPS

- Conditional Jump Instructions enable decision making in programs

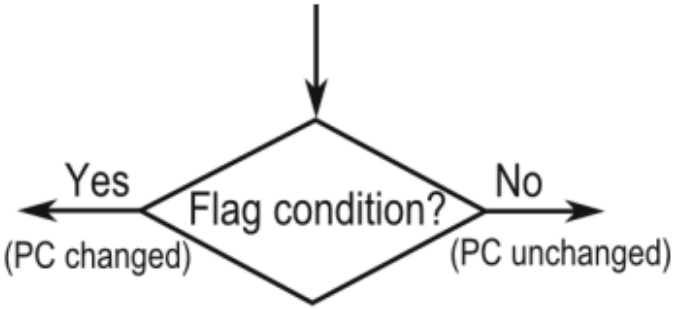


Fig. 3.26 Decision symbol associated to conditional jumps

Table 3.5 Conditional jumps

Mnemonics	Meaning	Mnemonics	Meaning
`jz`	Jump if zero (Z = 1)	`jn`	Jump if negative (N = 1)
`jnz`	Jump if not zero (Z = 0)	`jp`	Jump if positive (N = 0)
`jc`	Jump if carry (C = 1)	`jv`	Jump if overflow (V = 1)
`jnc`	Jump if no carry (C = 0)	`jnv`	Jump if not overflow (V = 0)

FLOWCHARTS AND JUMP INSTRUCTIONS

- Correspondence between some flowcharts constructs and register transfer notation (RTN)

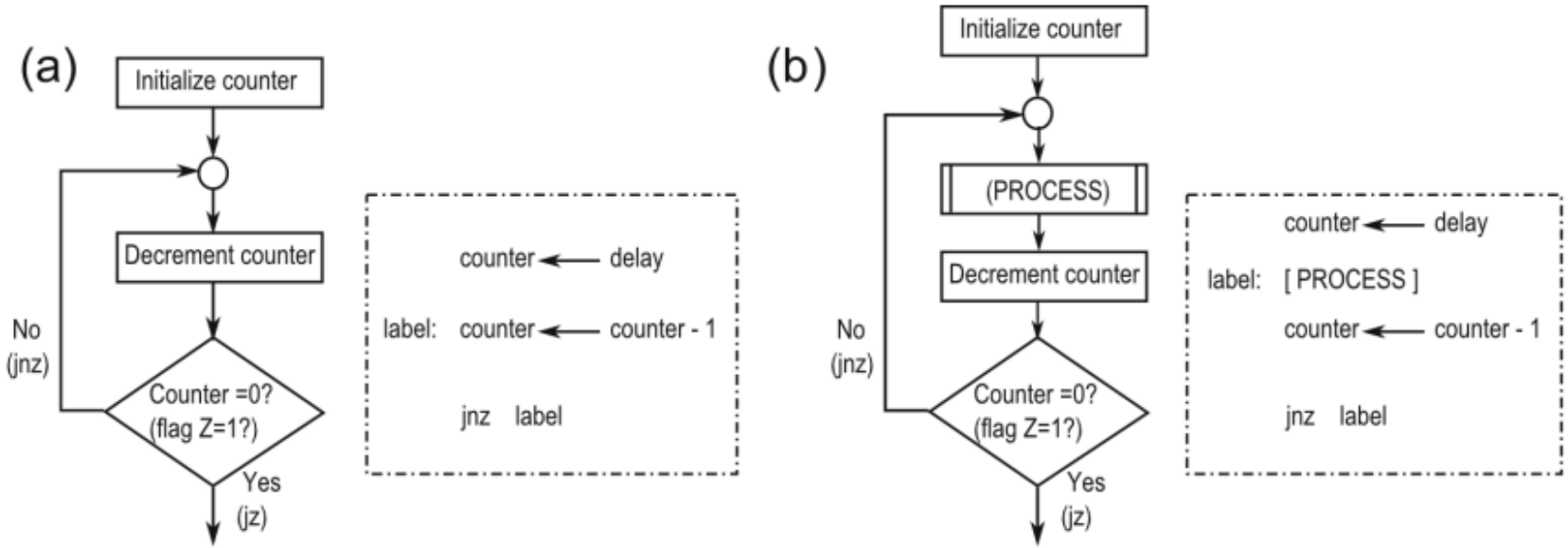


Fig. 3.27 Flow diagram and instruction skeleton associated to (**a**) delay loops and (**b**) iteration loops

THE STACK

- A portion of memory used to temporarily store data
  - Access through special register Stack Pointer (SP)
  - Last-in-First-out (LIFO) operation
  - Stack contents is volatile
- Stack Operations
  - PUSH: Places data on top of the stack
  - POP (or pull) : Retrieves data from the top of the stack
- Other instructions and events modifying the stack
  - Invoking and returning from a subroutine call
  - Responding and returning from an interrupt event

STACK OPERATIONS

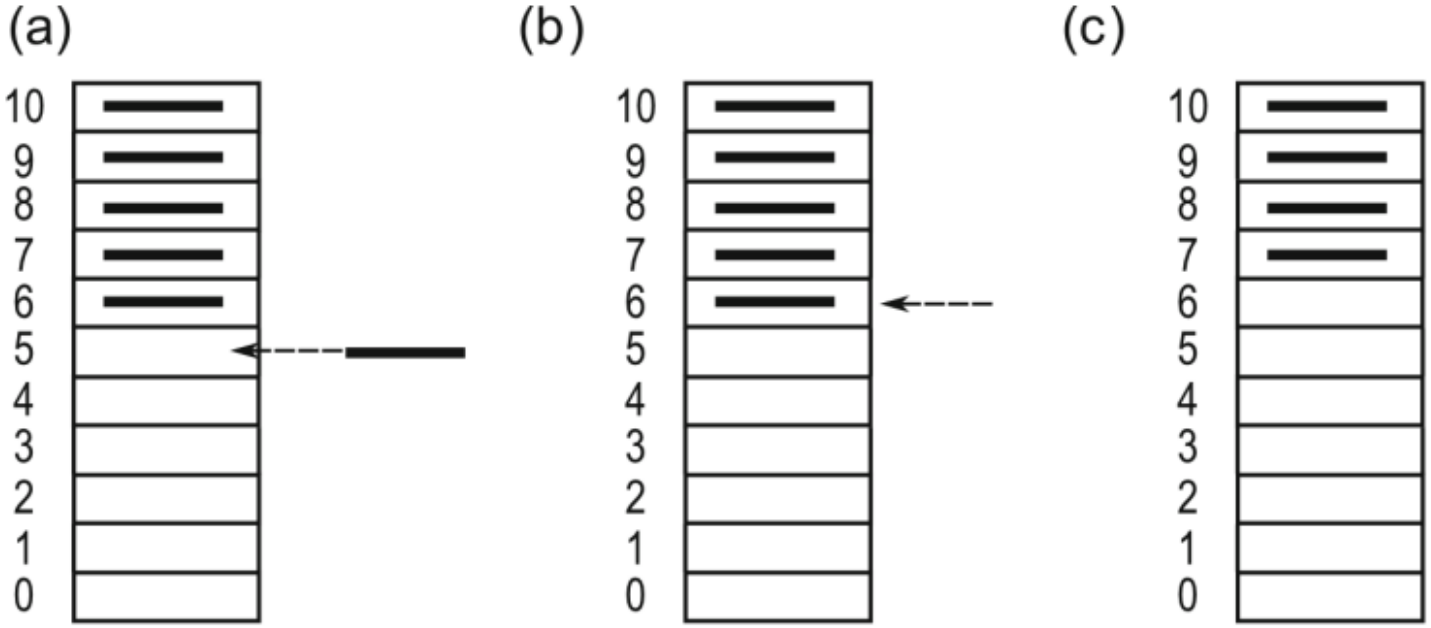


Fig. 3.28 Illustrating the stack operation. **a** TOS = 5 for pushing, **b** TOS = 6 for pulling, **c** After pulling, new TOS = 7

STEPS IN PUSH AND POP OPERATIONS

- Push
  - Update the stack pointer to point to the new TOS
  - Copy the operand to the new TOS
- Pop or Pull
  - Copy the contents in the actual TOS to the destination
  - Update the stack pointer to point to the new TOS
- Example: PUSH R9 and POP R9 (assume SP = 027Eh)

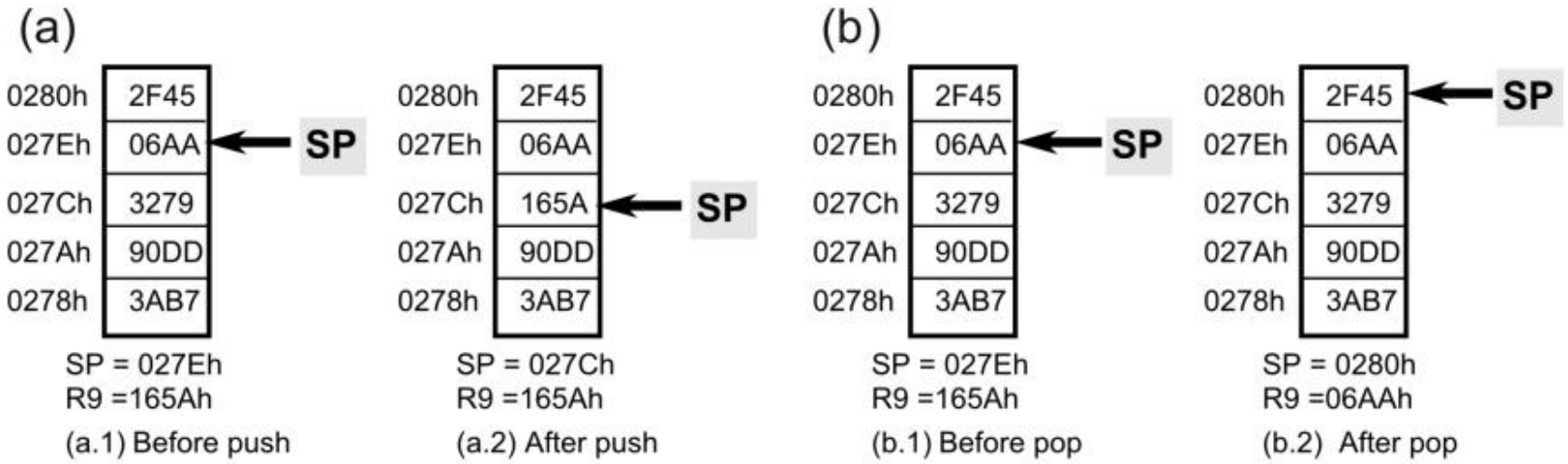


Fig. 3.29 The stack, SP register. **a** Push operation, **b** Pop operation

SAMPLE STACK OPERATIONS (1/3)

Store Before Update

CASE A: Pushing when SP points at TOS:
The operations are performed as follows:
1. Pushing a source:

$$(SP) \leftarrow src$$

$$SP \leftarrow SP - N$$

That is, it first stores the source and then updates SP.

2. Pulling into a destination:

$$SP \leftarrow SP + N$$

$$dst \leftarrow (SP)$$

That is, it first updates SP, and then retrieves the data.

Update Before Store

CASE B: Popping with SP pointing at TOS.
The operations are performed as follows:
1. When pushing a source:

$$SP \leftarrow SP - N$$

$$(SP) \leftarrow src$$

That is, it first updates SP and then stores the data.

2. When pulling into destination:

$$dst \leftarrow (SP)$$

$$SP \leftarrow SP + N$$

That is, it first retrieves the data and then updates the source.

STACK OPERATIONS (2/3)

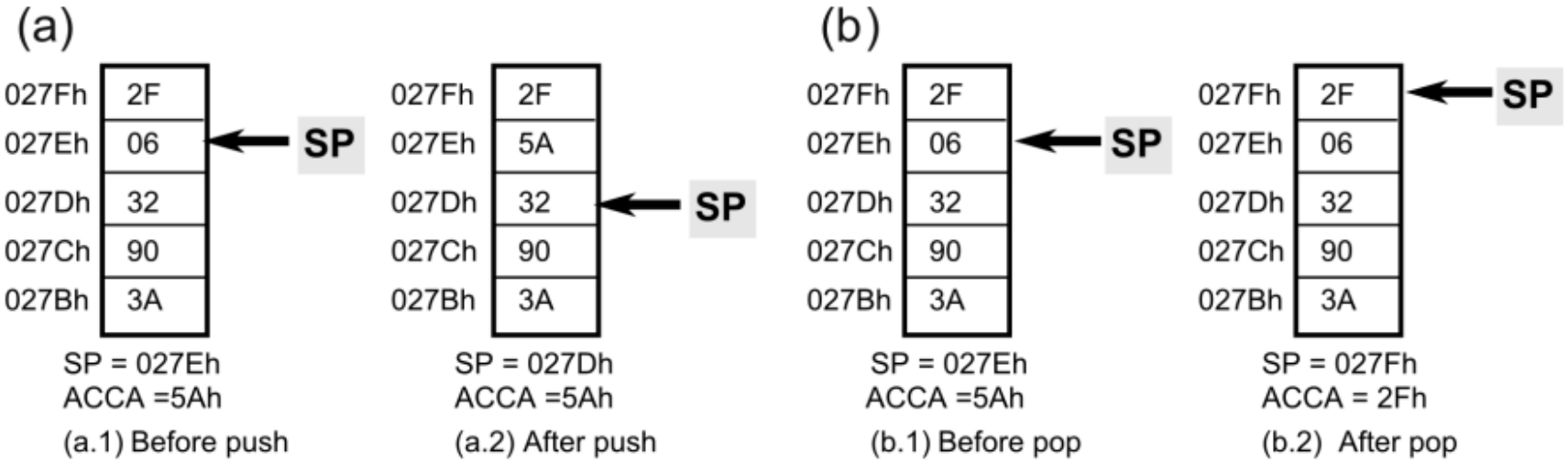


Fig. 3.30 Another example, SP register. **a** Push operation, **b** Pop operation

STACK OPERATIONS (3/3)

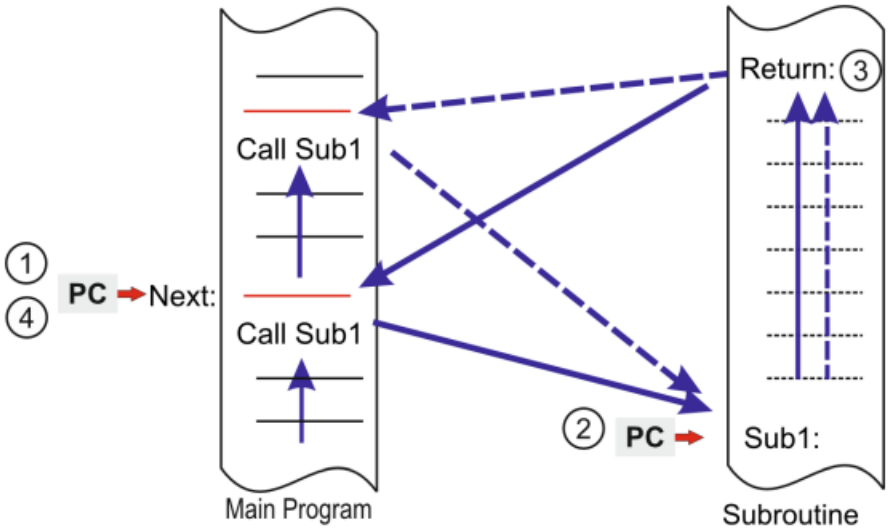


Fig. 3.31 Invoking a subroutine

1. **Function Call**
   Saves PC onto stack
2. **Function Execution**
   PC loaded with function address
3. **Executing the Return**
   Restore the PC from the stack
4. **Back at Main Program**
   Continue at instruction after "CALL"

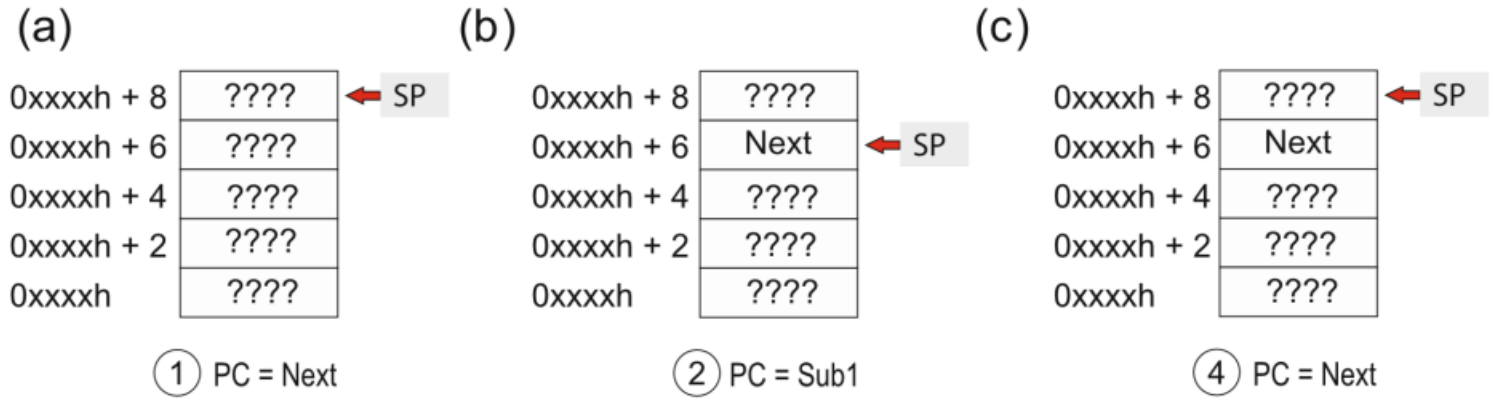


Fig. 3.32 Stack pointer use in the invocation and return of subroutines. **a** Stack before call, **b** Stack after call, **c** Stack after return

ADDRESSING MODES

- Addressing modes tell the CPU how to obtain the data needed to execute an instruction
- The data may be
  - Explicitly supplied with the instruction
  - Stored in a CPU register
  - Stored at a memory location
  - Stored in an I/O device register
- Implicit Addressing Mode
  - Operand is implicit to the instruction

BASIC ADDRESSING MODES

- Immediate Addressing Mode
  - Syntax: #Number
- Register Addressing Mode
  - Syntax: Rn
- Absolute or Direct Addressing Mode
  - Syntax: Number
- Indirect Addressing Mode
  - Syntax: @Rn
- Indexed Addressing Mode
  - Syntax: X(Rn)

SAMPLE ADDRESSING MODES

Table 3.6 MSP430 Instructions used for addressing mode examples

Assembly	RTN	Comment
`mov src,dest`	dest ← src	Copy or load source to destination
`add src,dest`	dest ← dest + src	Add source to destination
`sub src,dest`	dest ← dest − src	Subtract source from destination
`and src,dest`	dest ← dest .AND. src	Bitwise AND source to destination
`xor src,dest`	dest ← dest .XOR. src	Bitwise XOR source to destination
`cmp src,dest`	dest − src	Compare does not affect dest., only flags

ILLUSTRATING ADDRESSING MODES (1/2)

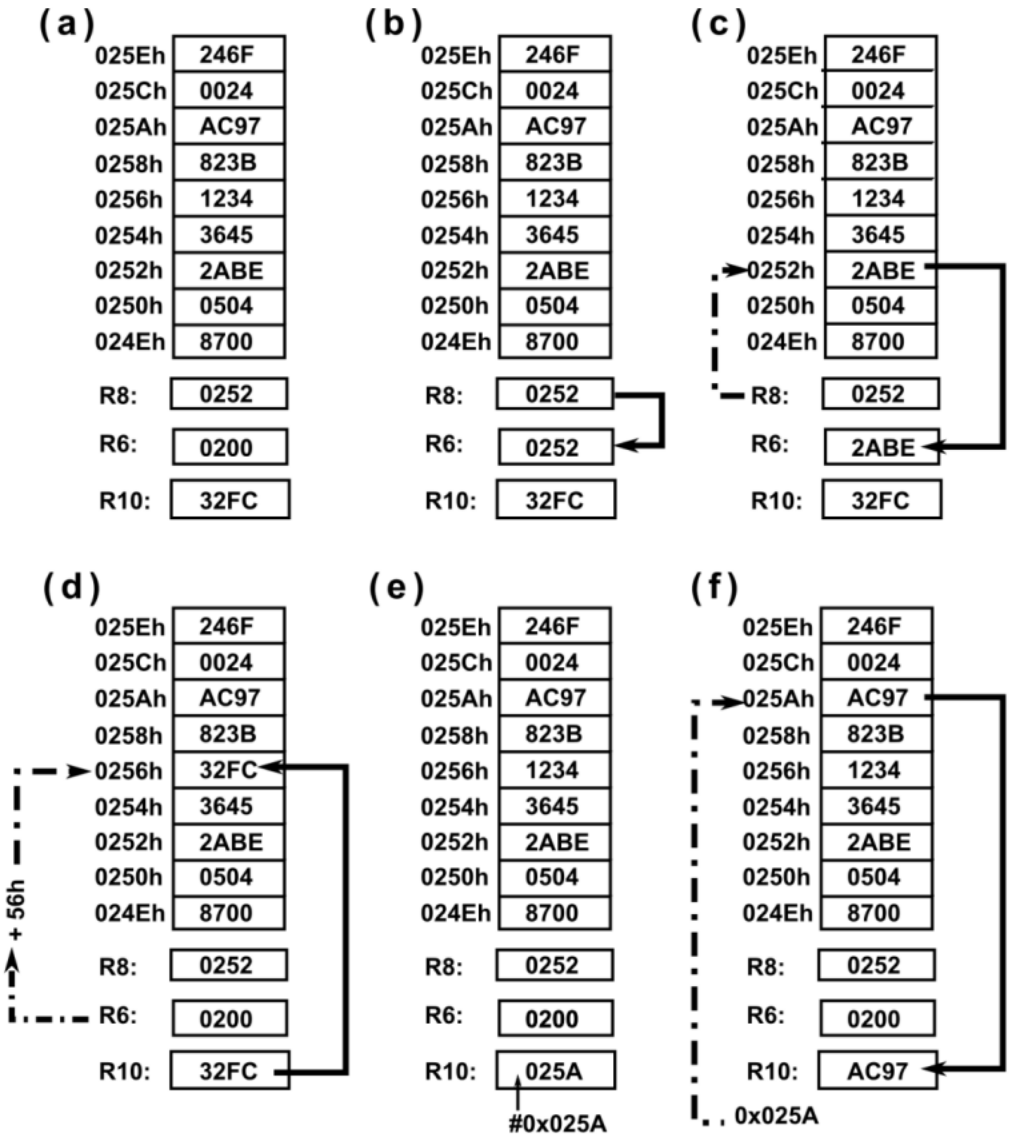


Fig. 3.33 Illustrating addressing modes.
(a) Intial condition,
(b) mov R8, R6,
(c) mov @R8, R6,
(d) mov R10, 56h(R6),
(e) mov #0x025A, R10,
(f) mov 0x025A, R10

ILLUSTRATING ADDRESSING MODES (2/2)

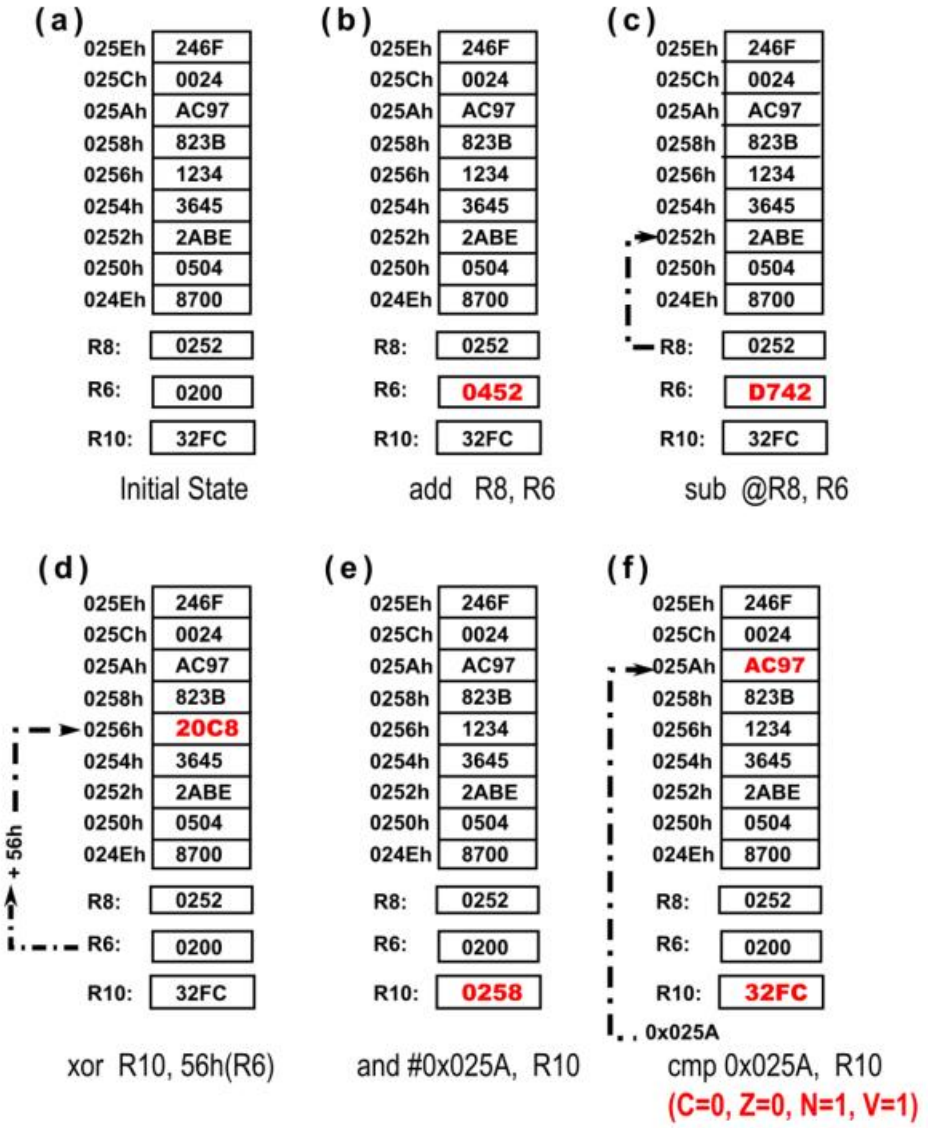


Fig. 3.34 Illustrating addressing modes with arithmetic-logic instructions

ORTHOGONAL CPU

- A CPU is said to be *orthogonal* if all its registers and addressing modes can be used as operands
  - Either source or destination
  - Excludes using the immediate addressing mode as destination
- Enables writing compact code
  - No need for intermediate storage instructions
- Require careful treatment
  - Special purpose registers can be adversely affected

DESIGN TIPS

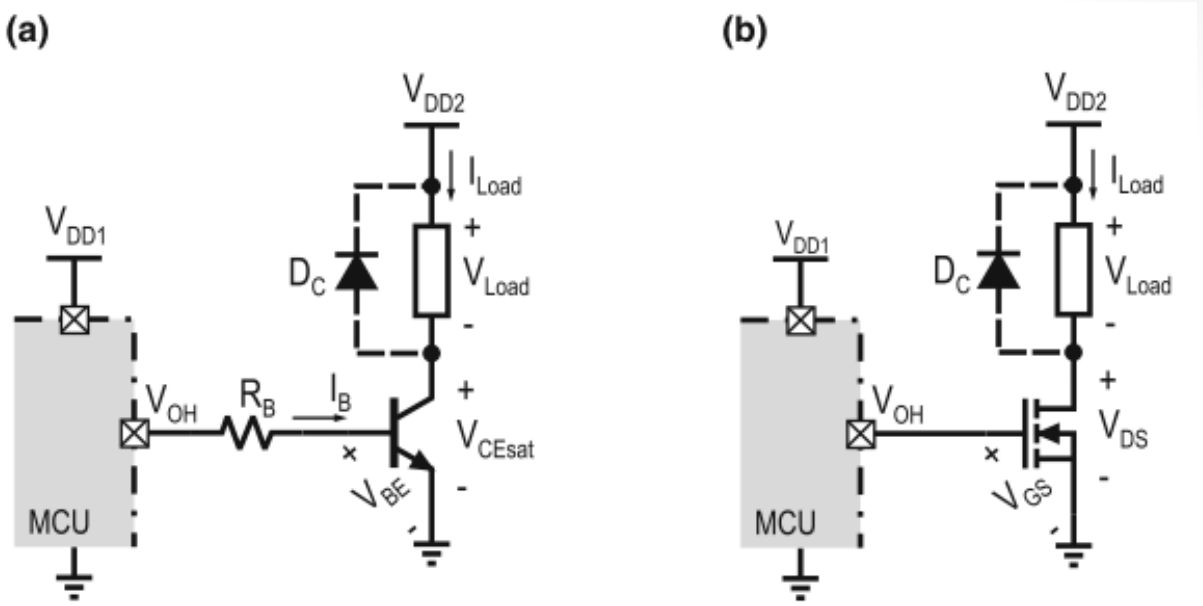


Fig. 8.50 Generalized topology of a transistor buffer using N-devices. **a** NPN buffer, **b** NMOS buffer

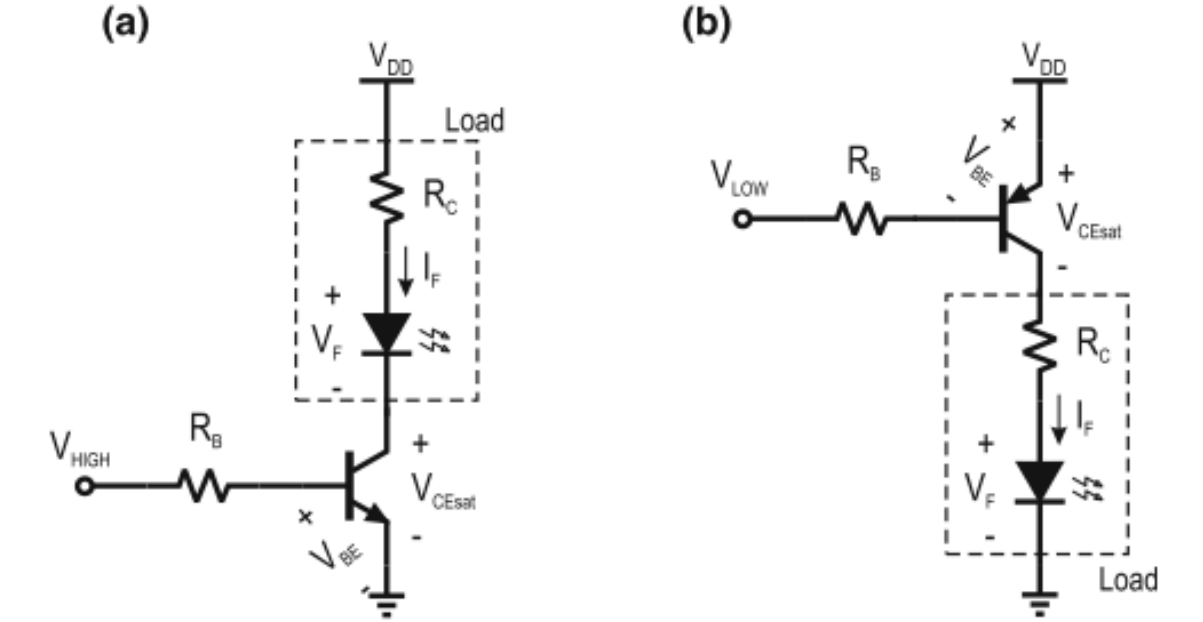


Fig. 8.51 Using bipolar transistors to buffer high-current LEDs. **a** NPN buffer, **b** PNP buffer

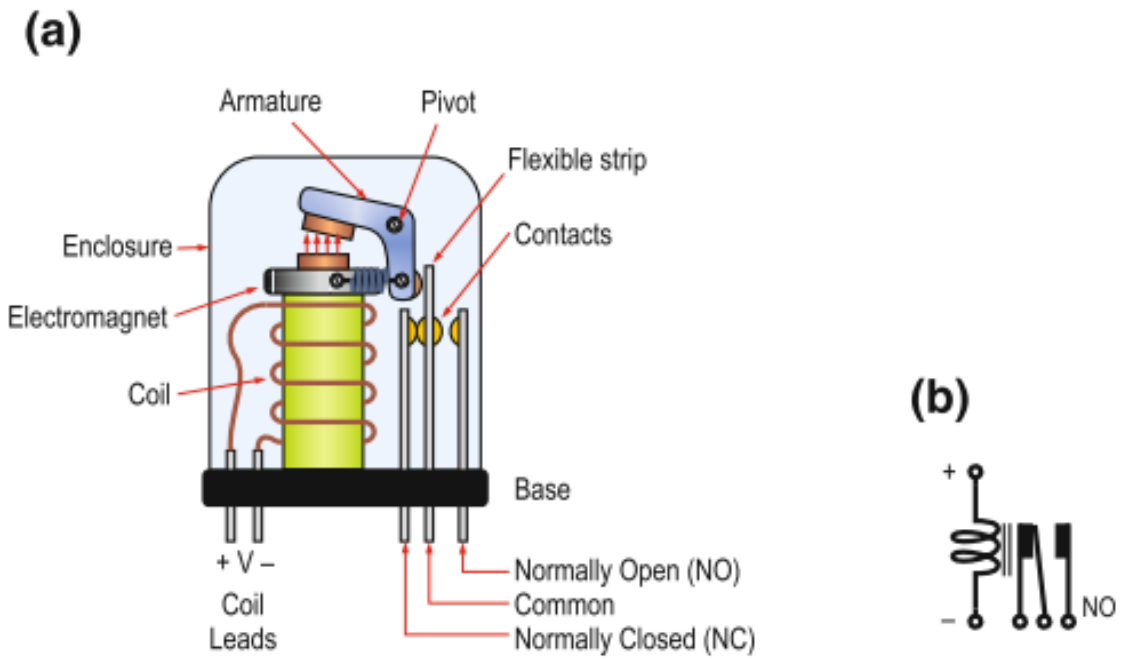
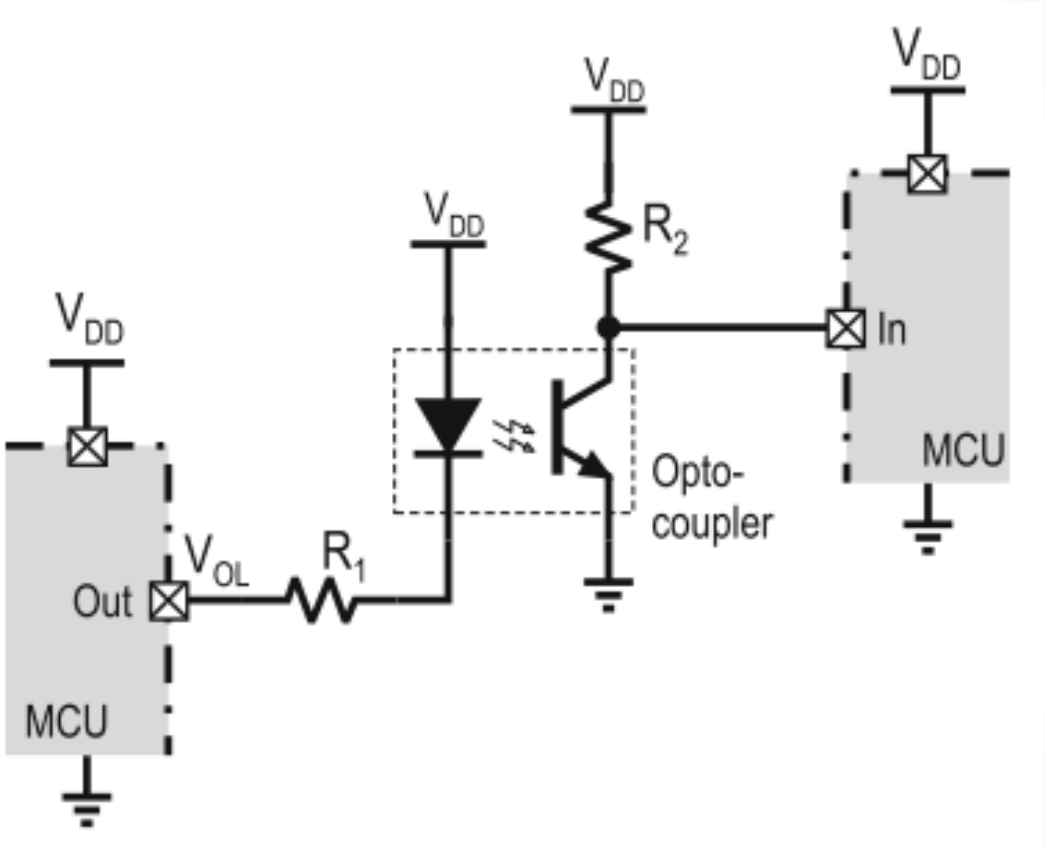


Fig. 8.55 Electromechanical relay with SPST NC/NO contacts, **a** Structure, **b** Symbol

Fig. 8.57 Interfacing an optocoupler to an MCU



- For more details, refer to:
  - Chapter 3,8 at **Introduction to Embedded Systems,** Springer 2014 by Manuel Jiménez et al.

- The lecture is available online at:
  - *http://bu.edu.eg/staff/ahmad.elbanna-courses*

- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg