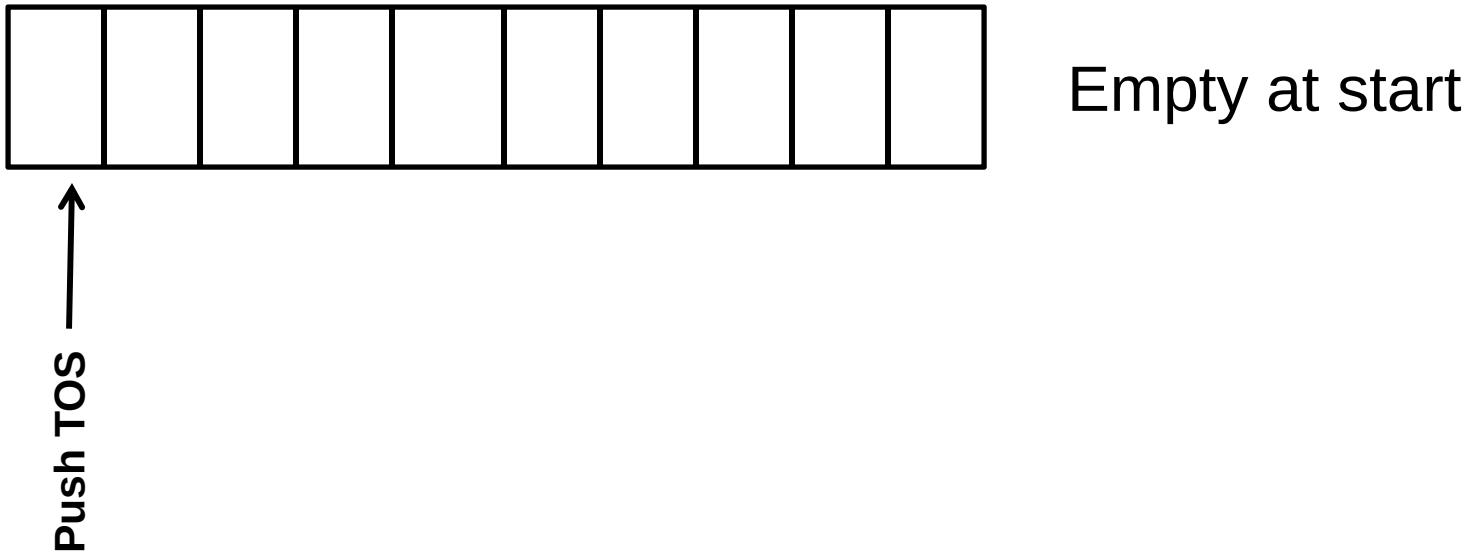
# STACK and Stack Pointer

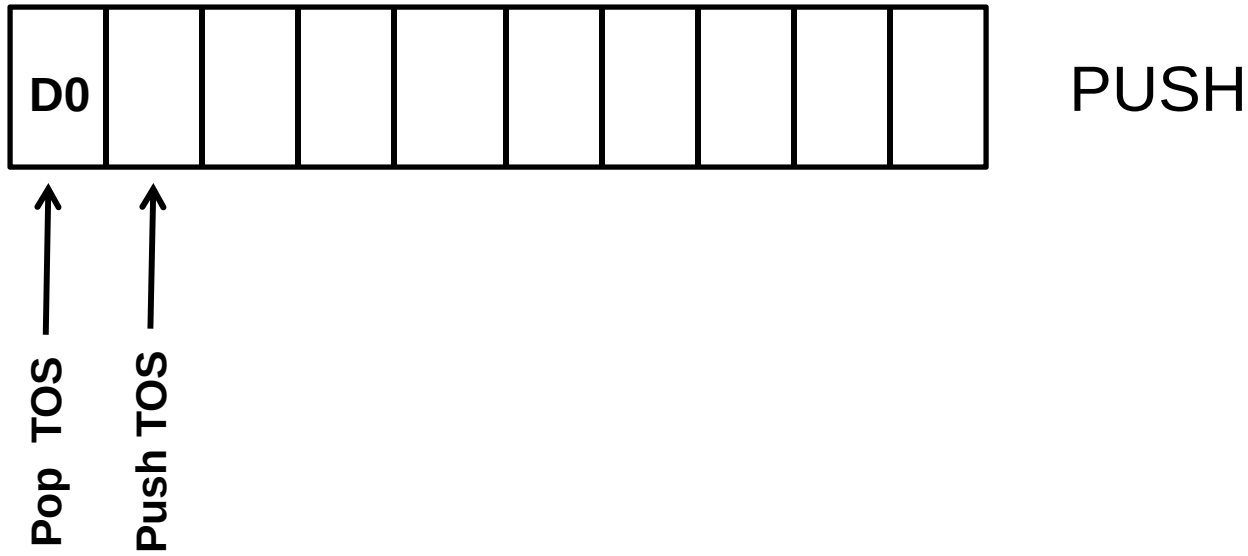
# Stack Definition and Characteristics

- Stack is a specialized memory segment which works in LIFO (Last In-First Out) mode. Managed by the Stack Pointer Register (SP)
  - Hardwired stack: physically defined, cannot change
  - Software defined: First address in stack defined by initialization of SP (by user or by compiler)
- Stack Operations:
  - **PUSH**: storing a new data at the next available location
  - **POP** or **PULL**: retrieving last data available from the sequence (to be stored in some destination)
  - Important Note: A retrieved data is not deleted, but cannot be retrieved again with a stack operation
- **Top of Stack (TOS)**: memory address used in the stack operation (different for push or pop)

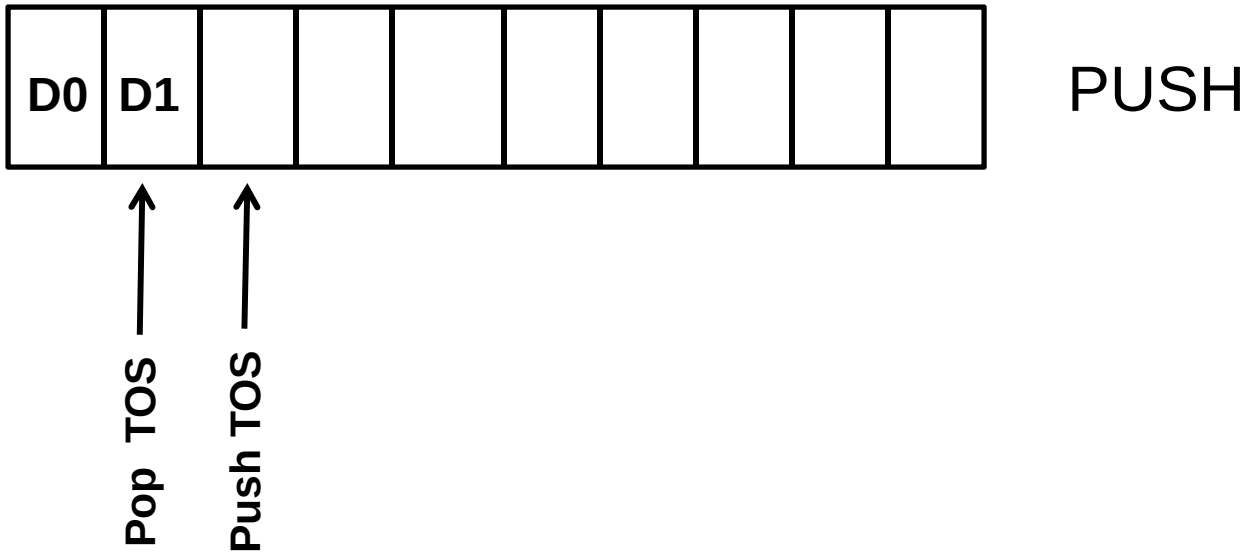
# Basics of stack operation



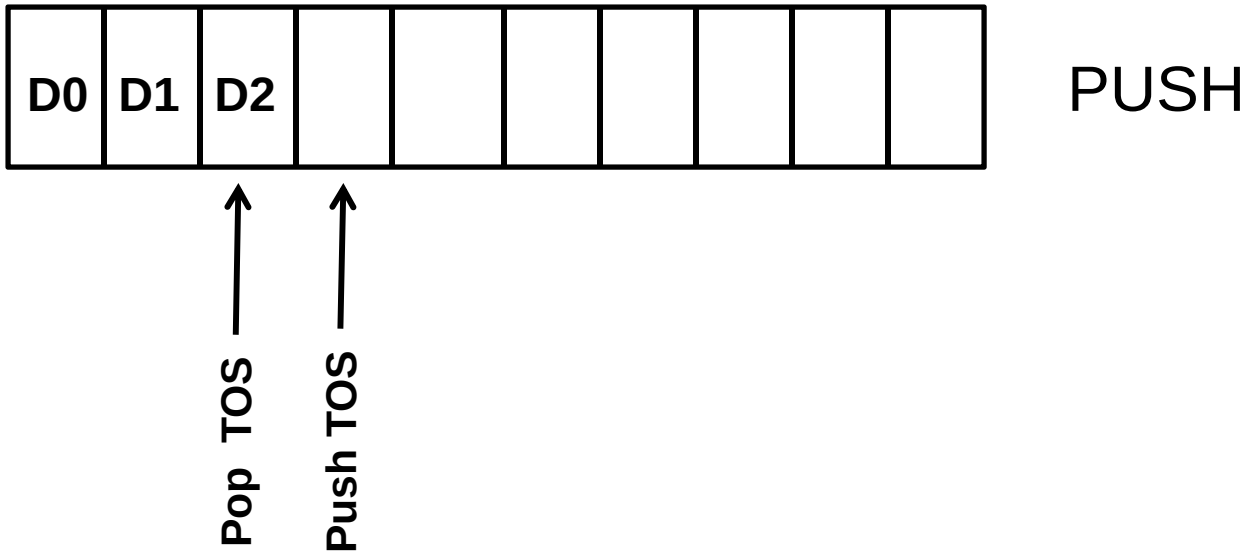
# Basics of stack operation



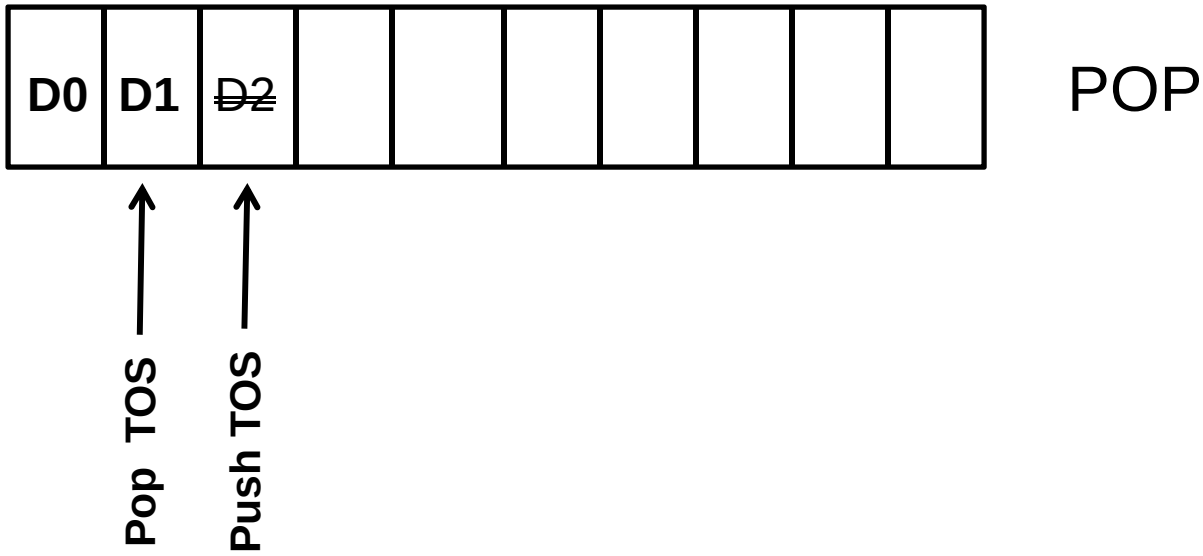
# Basics of stack operation



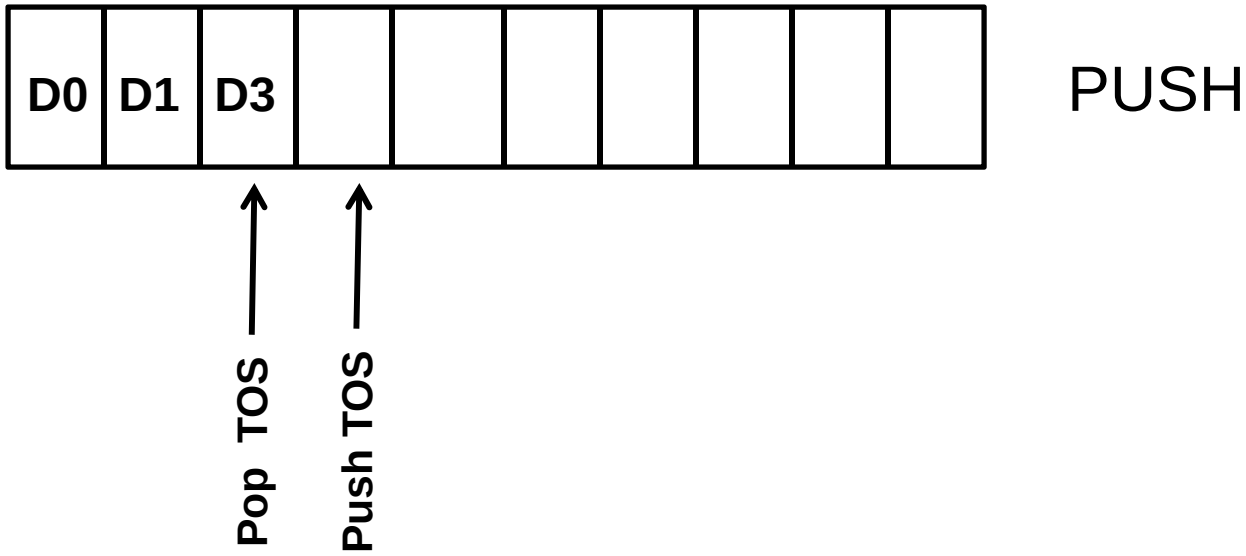
# Basics of stack operation



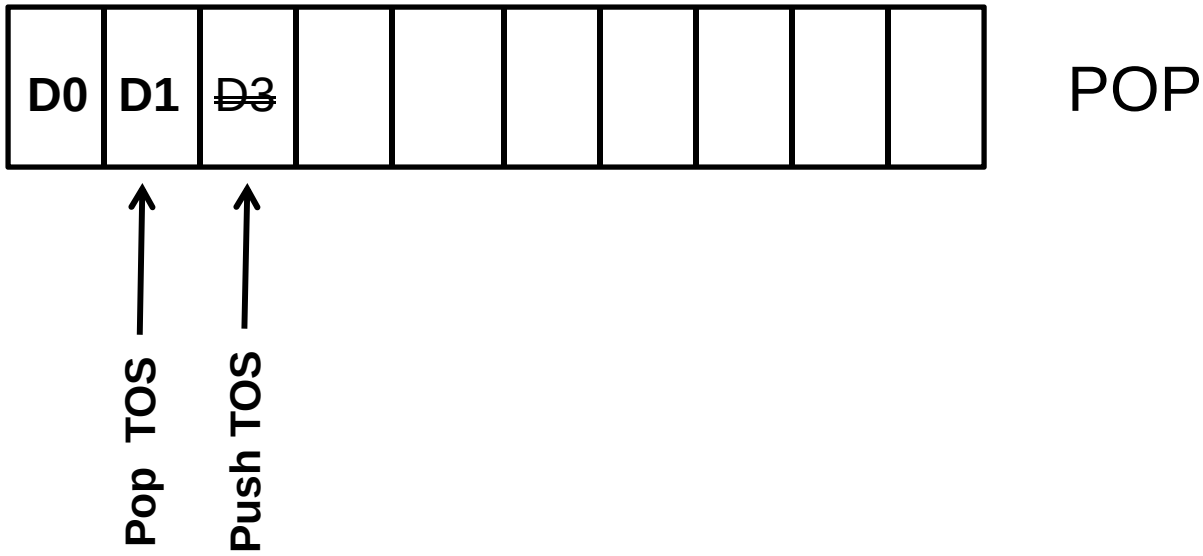
# Basics of stack operation



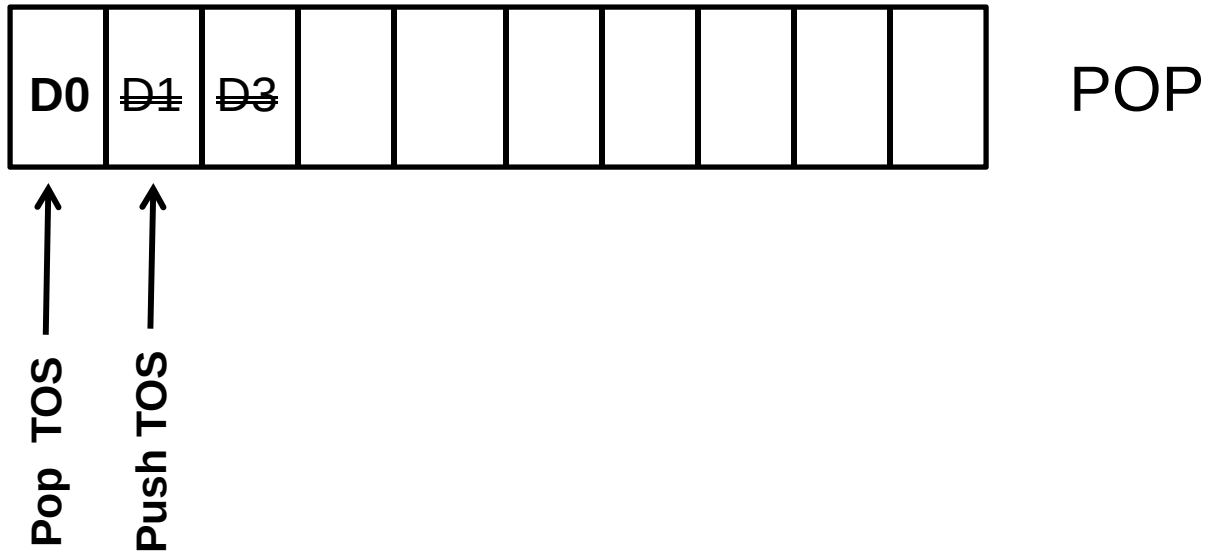
# Basics of stack operation



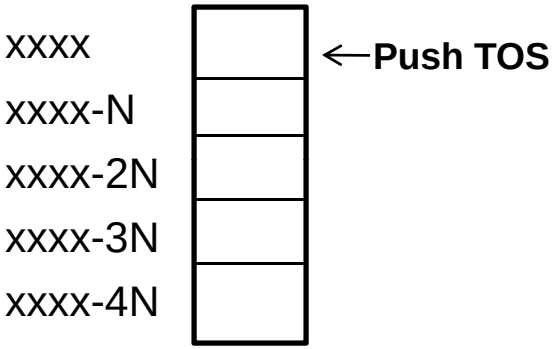
# Basics of stack operation



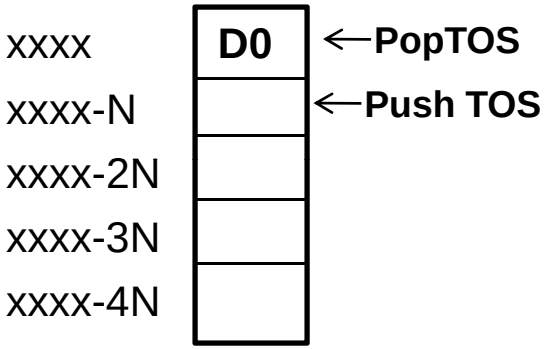
# Basics of stack operation



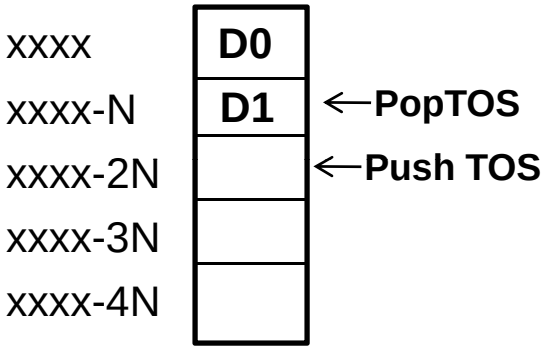
# Software Defined Stack Grows Downwards



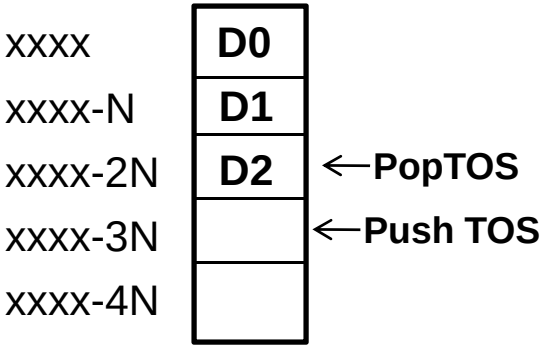

Empty



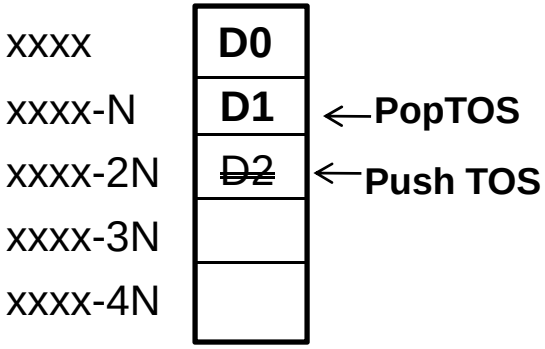

After a push



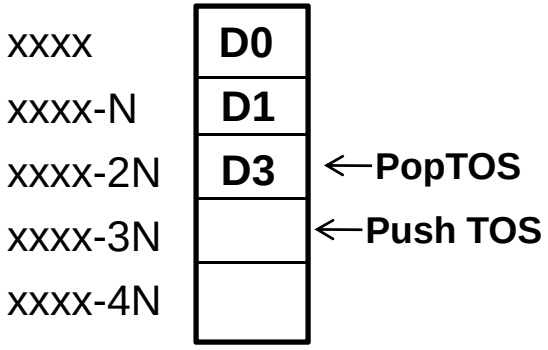

After a push




After a push




After a pop



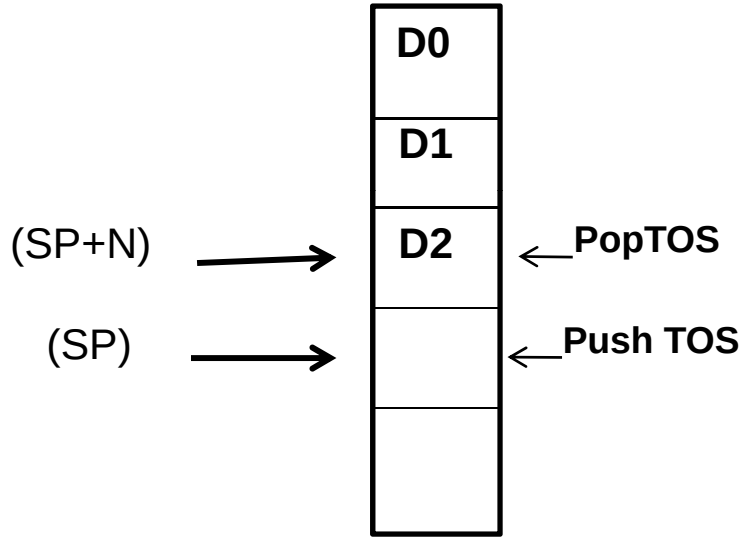

After a push

# Stack and Stack Pointer

- The Stack Pointer contents is an address associated to the stack operation
  - The contents of SP is sometimes called Top-of-Stack
- Since contents is unique, and two addresses are associated to the TOS, there are 2 possibilities:
  - SP contains the PUSH-TOS (Example: Freescale)
  - SP contains the POP TOS (Example: MSP430

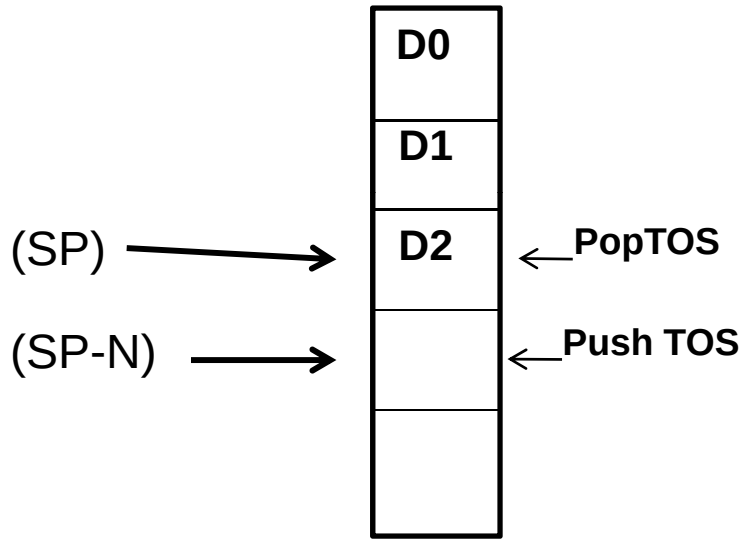
# SP points to PUSH TOS

- To do a PUSH:
  - 1. Store (SP) ← Data
  - 2. Update SP ← SP - N
- To do a POP:
  - 1. Update SP←SP+N
  - 2. Retrieve Dest←(SP)
- These steps are done automatically by CPU.



# SP points to POP TOS

- To do a PUSH:
  - 1. Update SP ← SP -
  - 2. NStore (SP) ← Data
- To do a POP:
  - 1. Retrieve Dest←(SP)
  - 2. Update SP←SP+N
- These steps are done automatically by CPU.



# INSTRUCTION SET

1. Machine instructions

# Machine Instruction

- System “understands” only 0’s and 1’s.
- A set of Word(s) processed in the instruction register becomes an instruction
  - The instruction may consists of one or several words
  - **The first word is the <u>INSTRUCTION WORD</u>**
- The size of instruction words may be different or not, depending on CPU, IR, and if Harvard or Von Neumman architecture.

# Instruction Word Structure

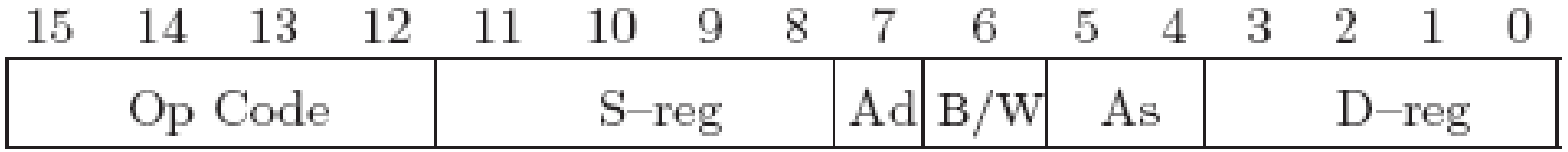
- **OpCode**: (Operating Code)Field of bits in the instruction word that indicates what operation is done
- Operands and Addressing Modes Fields**:** Field(s) of bits indicating which operands are used in the transaction and where to find data.
  - Operands may be implicitly included (implicit operand)
- The complete structure is CPU dependent.

# EXAMPLE: MSP430 INSTRUCTIONS (1/4 )
# 1. General facts

- Instructions are divided in four groups
  - Two-operand instructions (*source and destination*)
  - One-operand instructions (*source OR destination*)
  - Jump instructions (Operand is an offset) :implicit operand PC
  - Return from Interrupt (RETI) instruction: Implicit operands PC and SR: 1300h
- Note: TI user guides classify RETI as a one operand instruction

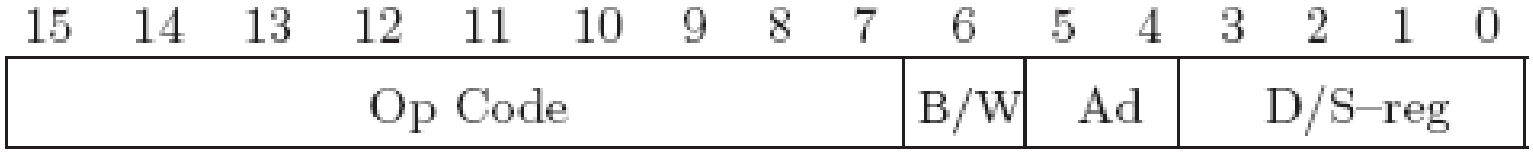
# EXAMPLE: MSP430 INSTRUCTIONS (2/4 )
# 2. Two Operand Instructions

- Bits 15-12: OP-CODE (4 to F)
- Bits 11-8: Source info
- Bits 7-4: Operands and addressing modes
- Bits 3-0: Destination info



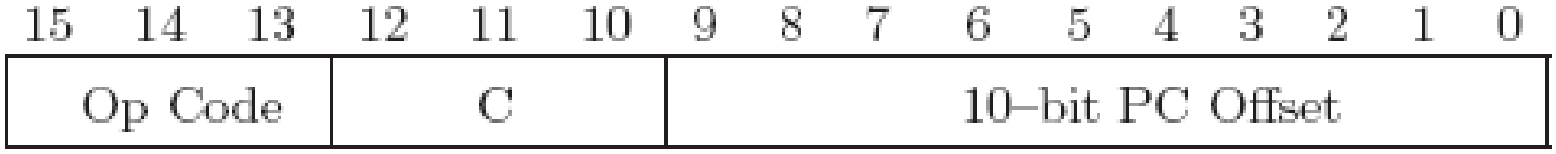
# EXAMPLE: MSP430 INSTRUCTIONS (3/4 )
# 3. Single Operand Instructions

- Bits 15-7: OP-CODE (Most significant nibble is 1)
- Bit 6: (W/B) 0 for word size operand, 1 for byte size
- Bits 5-4: Addressing mode for operad
- Bits 3-0: Operand info



# EXAMPLE: MSP430 INSTRUCTIONS (4/ 4)
# 4. Jump Instructions

- Bits 15-13: OP-CODE
- Bit 12-10: Condition statement (8 conditions)
- Bits 9-0: 10-bit signed offset for PC (-512 to 511)
- Notes:
  - Most significant nibble is 2 or 3
  - To effectuate jump, PC ← PC + 2 (Offset)
    - Maximum jump size 1K: -1,024 to +1,022



# Instruction Set

2. Assembly language

# Assembly Language Characteristics

- Instructions in assembly language are “human friendly” notations for machine instructions
  - **Each assembly language instruction corresponds to one machine instruction only, and viceversa**
- Components of instruction:
  - **Mnemonics**: associated to OpCode (and other information in instruction)
  - **Operands**: Written in a special syntax form called Addressing Mode
- IMPORTANT: Assembly is proper to microcontroller family.

# Mnemonics examples for MSP430 (1a/3)
# Dual Operand Instructions

Table B.1: Opcodes for dual-operand instructions

| Hex Code (Bits 15-12) | Instruction | Hex Code (Bits 15-12) | Instruction |
|---|---|---|---|
| 4 | `mov.w, mov.b` | A | `dadd.w, dadd.b` |
| 5 | `add.w, add.b` | B | `bit.w, bit.b` |
| 6 | `addc.w,addc.b` | C | `bic.w, bic.b` |
| 7 | `subc.w, subc.b` | D | `bis.w, bis.b` |
| 8 | `sub.w, sub.b` | E | `xor.w, xor.b` |
| 9 | `cmp.w, cmp.b` | F | `and.w, and.b` |

**Notes**:

1. **** .w means that bit 6 (B/W) is 0. (Word size operands)
2. ****.b means that bit 6 (B/W) is 1. (Byte size operands)
3. Suffix w may be omitted (mov.w = mov)

# Mnemonics examples for MSP430 (2/3) Single Operand Instructions and **RETI**

Table B.2: OpCodes for single operand instructions

| Bits 15-12 | Bits 11-7 | B/W bit | instruction |
|---|---|---|---|
| 0001 | 00000 | 0,1 | `rrc, rrc.b` |
| 0001 | 00001 | 0 | `swpb*` |
| 0001 | 00010 | 0,1 | `rra, rra.b` |
| 0001 | 00011 | 0 | `sxt*` |
| 0001 | 00100 | 0,1 | `push, push.b` |
| 0001 | 00101 | 0 | `call*` |

* Word instruction only.

Instruction **reti** has no explicit operands, and only a 16-bit opcode: **1300**

# Mnemonics for MSP430 Instructions (1/3)
# Jump Instructions

Table B.3: OpCodes for jump instructions

| Bits 15-13 | Bits 12-10 | instruction |
|---|---|---|
| 001 | 000 | jnz/jneq |
| 001 | 001 | jz/jeq |
| 001 | 010 | jnc/jlo |
| 001 | 011 | jc/jhs |
| 001 | 100 | jn |
| 001 | 101 | jge |
| 001 | 110 | jl |
| 001 | 111 | jmp |

# Mnemonics for MSP430 (4/6)
# “Reading”

| Mnemonics | Reading | Mnemonics | Reading |
|---|---|---|---|
| **mov** | move | **dadd** | Decimal (BCD) addition with carry |
| **add** | add | **bit** | Bit Test |
| **addc** | add with carry | **bic** | Bit Clear |
| **subc/ sbb** | subtract with borrow (carry) | **bis** | Bit Set |
| **sub** | subtract | **xor** | X-OR |
| **cmp** | compare | **and** | AND |

# Mnemonics for MSP430 (4/6) "Reading"

| Mnemonics | Reading | Mnemonics | Reading |
|---|---|---|---|
| **rrc** | rotate right through carry | **jnz/jneq** | Jump if not zero/ if not equal |
| **swpb** | swap bytes | **jz/jeq** | Jump if zero/ if equal |
| **rra** | roll right arithmetically* | **jnc/jlo** | jump if no carry/if lower than |
| **sxt** | sign extend low byte | **jc/jhs** | jump if carry/ if higher or same |
| **push** | push | **jn** | jump if negative |
| **call** | call | **jge** | jump if greater or equal |
| **reti** | Return from interrupt | **jl** | jump if less than |
| | | **jmp** | jump unconditionally |

Note: rra is also read as "rotate right arithmetically, but in fact it does not rotate. It is also a 'shift right arithmetically'

# Types of Instructions (1/2)

- **Data Transfer**: copy data from a source onto a destination [ dest ← src ]
- **Operations**: Combine data from two operands according to an operation rule

  [dest← src1*src2 or dest← dest*src ]

  - Arithmetic and Logic
  - Bitwise Logic: Bit by bit [dest(j)←dest(j) * src(j) ]
  - Compare and testing: affect flags but not operands
  - Rotate and shift (roll): Displace bits internal to operand

# Types of Instructions (2/2)

- **Program flow or Program Control:** Change the default address of next instruction

  [PC← NewAddress]

  – Jump or branch instructions

  – Subroutine and interrupt handling instructions

- **Miscellaneous**: not in previous categories

  – Example: No Operation – an instruction that actually does nothing but causes a delay.