Introduction to Computing Systems
from bits & gates to C & beyond

# Chapter 5

## The LC-3 Instruction Set Architecture

- *ISA Overview*
- *Operate instructions*
- *Data Movement instructions*
- *Control Instructions*
- *LC-3 data path*

# LC-3 ISA Overview

- **Memory organization**
  - Address space: $2^{16}$ = 64k locations
  - Addressability: Word
  - Word size = 2 bytes
    - *=> total memory = 64k x 2 = 128 kbytes*
- **Registers**
  - 8 x 16 bit General Purpose Registers: R0 - R7
  - 3 x 1 bit Condition Code Registers: N, Z, P
  - Some special purpose registers (more later)
- **Instructions**
  - 16 bit instructions, with 4 bit opcodes
  - Native Data Type: only 2's complement integer
  - Addressing Modes: Immediate, Register, Direct, Indirect & Base+Offset

# LC-3 Instructions

- **Operate**
  - **Manipulate data directly - arithmetic or bitwise logic**
    - ***ADD, AND, NOT***
- **Data Movement**
  - **Move data between memory and registers**
    - ***LD, LDI, LDR, LEA, ST, STI, STR***
- **Control**
  - **Change the sequence of instruction execution**
    - ***BR, JMP/RET, JSR/JSRR, TRAP, RTI***

# Instruction Construction

- **Two main parts**
  - Opcode: specifies what the instruction does.
  - Operand(s): what the instruction acts on.
  - Instruction sets can be complex or simple (CISC, RISC), single-word or multi-word.

- **LC-3**
  - Single word (16 bit) instructions.
  - 4-bit opcode => 16 instructions (very simple set!)
  - remaining 12 bits specify operand(s), according to the addressing mode proper to each instruction.

# LC 3 Instructions

- **LC-3 Instruction word: 16 bits**
  - **Opcode**
    - ***IR[15:12]: 4 bits allow 16 instructions***
    - ***specifies the instruction to be executed***
  - **Operands**
    - ***IR[11:0] contains specifications for:***
      - **Immediate value: 5 bits**
      - **Registers: 8 GPRs (i.e. require 3 bits for addressing)**
      - **Address Generation bits: "Offset" (11 or 9 or 6 bits - more later)**
- **Examples**

| ADD | DR, | SR1, | SR2 | ; DR ← (SR1) + (SR2) |
|---|---|---|---|---|
| [15:12] | [11:9] | [8:6] | [2:0] | |

| LDR | DR, | BaseR, | Offset | ; DR ← Mem[ (BaseR) + Offset6 ] |
|---|---|---|---|---|
| [15:12] | [11:9] | [8:6] | [5:0] | |

# Addressing Modes

**Note: the effective address (ea) is the memory location of the operand**

- **The LC-3 supports five addressing modes:**
  - **the operand is located:**
    - ***in the instruction itself (immediate)***
    - ***in a register***
    - ***in memory:***
      - **the ea is encoded in the instruction (direct, or PC-relative)**
      - **a pointer to the ea is encoded in the instruction (indirect)**
      - **a pointer to the ea is stored in a register (relative, or base+offset)**

# Operate Instructions

- **NOT (unary operator)**
  - destination register in IR[11:9] and a single source register in IR[8:6].
  - bits IR[5:0] are all 1s.

- **ADD & AND (binary operators)**
  - destination register in IR[11:9], one source register in IR[8:6]
  - other source:
    - *immediate addressing mode:*
      - if bit IR[5] = 1, bits IR[4:0] specify the other source number directly, as a 5 bit 2's complement integer, which is sign extended (SEXT) to 16 bits.
    - *register addressing mode:*
      - if bit IR[5] = 0, bits IR[2:0] specify a register for the second source
      - bits IR[4:3] = 0

# Immediate & Register Operands

- **Immediate**

| opcode | operands | | | |
|---|---|---|---|---|
| [15:12] | [11:9] | [8:6] | [5] | [4:0] |
| ADD | DR | SR1 | 1 | imm |

  - **If bit 5 = 1, the value in IR[4:0] ("immediate") is *sign extended (SEXT)* to 16 bits and added to the contents of the source register SR1 (IR[8:6]).**

- **Register**

| opcode | operands | | | | |
|---|---|---|---|---|---|
| [15:12] | [11:9] | [8:6] | [5] | | [2:0] |
| ADD | DR | SR1 | 0 | | SR2 |

  - **if bit 5 = 0, the contents of source register SR2 (IR[2:0]) are added to the contents of source register SR1 (IR[8:6]).**
  - **In both cases, the result goes to the destination register DR (IR[11:9]).**

# NOT: Bitwise Logical NOT

- **Assembler Inst.**

  NOT DR, SR ; DR <= NOT (SR)

- **Encoding**

  1001 DR SR 111111

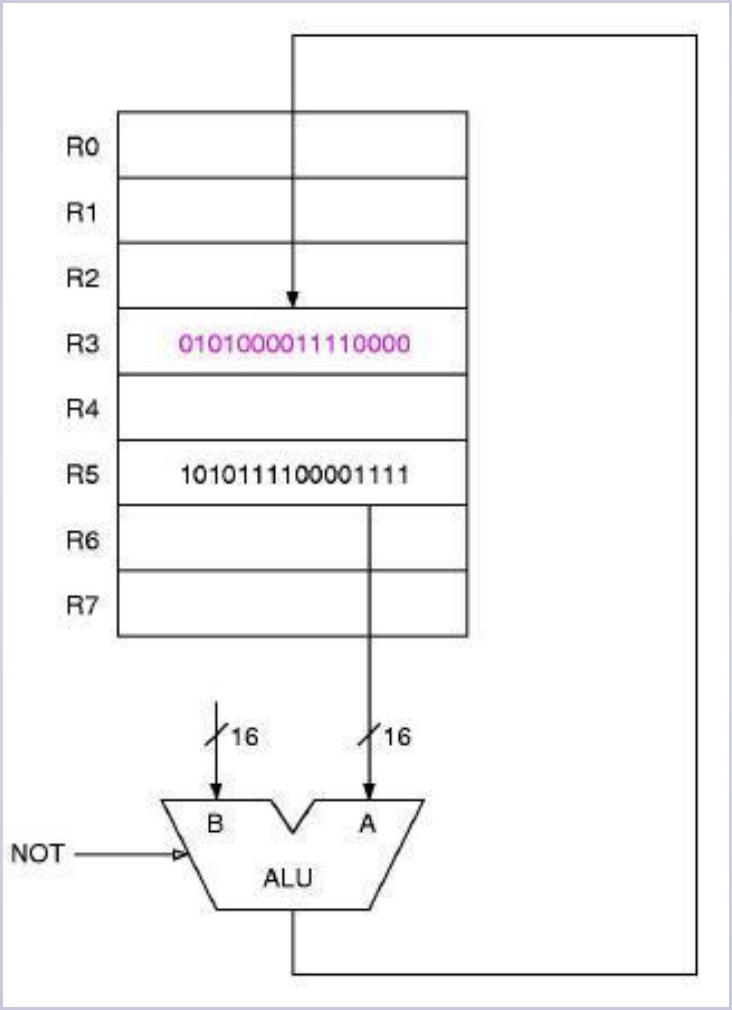
- **Example**

  NOT R2, R6

  - **Note: Condition codes are set.**

# NOT data path

**NOT R3, R5**



# ADD: Two's complement 16-bit Addition

- **Assembler Instruction**

  ADD DR, SR1, SR2 ; DR <= (SR1) + (SR2) *(register addressing)*
  ADD DR, SR1, imm5 ; DR <= (SR1) + Sext(imm5) *(immediate addressing)*

- **Encoding**

  0001 DR SR1 0 00 SR2
  0001 DR SR1 1 imm5

- **Examples**

  ADD R1, R4, R5
  ADD R1, R4, # -2

  - **Note: Condition codes are set**

# AND: Bitwise Logical AND

- **Assembler Instruction**

  AND DR, SR1, SR2 ; DR <= (SR1) AND (SR2)
  AND DR, SR1, imm5 ; DR <= (SR1) AND Sext(imm5)

- **Encoding**

  0101 DR SR1 0 00 SR2
  0101 DR SR1 1 imm5

- **Examples**

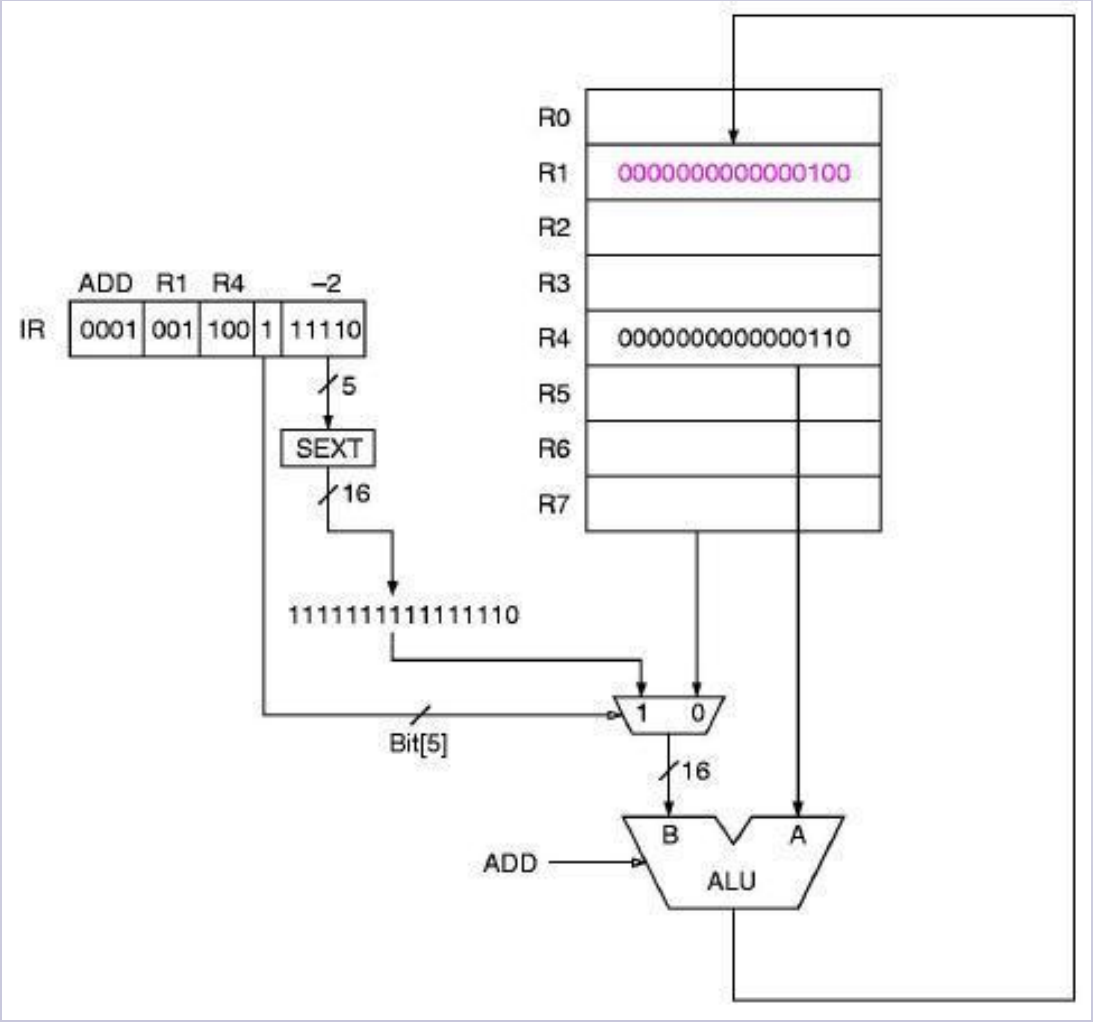
  AND R2, R3, R6
  AND R2, R2, #0 ; Clear R2 to 0

  **Question:** ***if the immediate value is only 6 bits, how can it mask the whole of R2?***

  - **Note: Condition codes are set.**

# ADD data path

## ADD R1, R4, # -2



# Data Movement Instructions - 1

- **Move Data**
  - from register to memory => store
    - *nominated register is Source*
  - from memory to register => load
    - *nominated register is Destination*
  - The LC-3 cannot move data from memory to memory
  - also to/from I/O devices (later)
- **LC-3 Load/Store Instructions**
  - LD, LDI, LDR, LEA, ST, STI, STR
  - Format:


***Note: The field 'PCoffset9' in the following slides is IR[ 8:0 ]***

# Data Movement Instructions - 2

- **LC-3 Load/Store Addressing modes:**
  - ***immediate: LEA***

    **No Effective Address (EA) calculation; the Sign Extended Addr. Generator is added to the current value of the Program Counter - i.e.**

    **DR <= (PC) + SEXT( IR[8:0] )**
  - ***direct or PC-Relative: LD & ST***

    **The EA is the Sign Extended Addr. Generator added to the current value of the Program Counter - i.e.**

    **EA = (PC) + SEXT( IR[8:0] )**

    **DR <= Mem[ (PC) + SEXT( IR[8:0] ) ]**
  - ***indirect: LDI & SDI***

    **EA = Mem[ (PC) + SEXT( IR[8:0] ) ]**

    **DR <= Mem[Mem[ (PC) + SEXT( IR[8:0] ) ] ]**
  - ***base+offset: LDR & STR (BaseReg is specified by IR[8:6])***

    **EA = BaseReg + SEXT( IR[5:0] )**

    **DR <= Mem[ (BaseReg) + SEXT( IR[5:0] ) ]**

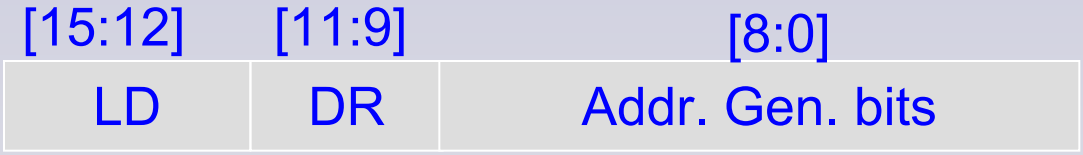




Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# Memory Addressing Modes

- **Direct addressing (PC-Relative)**




  - **effective address = (PC) + SEXT( IR[8:0] )**
  - **operand location must be within approx. 256 locations of the instruction**
    - ***actually between +256 and -255 locations of the instruction being executed (why?)***

# Memory Addressing Modes - 2

- **Indirect addressing**


  - **Same initial mechanism as direct mode (i.e. PC-Relative), but the calculated memory location now contains the *address of the operand,* (i.e. the ea is indirect):**

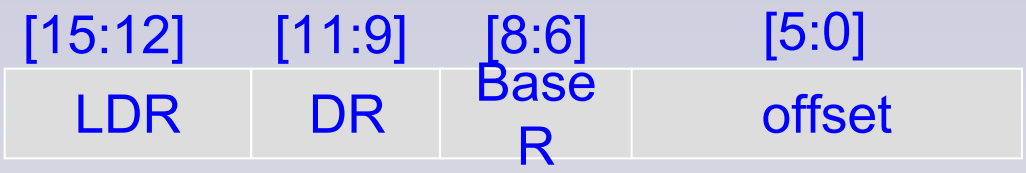
    **pointer address = (PC) + SEXT( IR[8:0] )**

    **effective address = Mem[ (PC) + SEXT( IR[8:0] ) ]**

    - ***Note that the memory has to be accessed twice to get the actual operand.***

# Memory Addressing Modes - 3

- **Base+Offset addressing**




  - **effective address = (BaseRegister) + offset**
    - ***sign extend (SEXT) the 6 bit offset ([5:0]) to 16 bits***
    - ***add it to the contents of the Base Register ([8:6])***

  - **differences other addressing modes:**
    - ***base+offset field is 6 bits, PC-Relative offset field is 9 bits.***
    - ***base+offset can access any location in memory, PC-Relative offset only within +/- 256 locations of the instruction.***
    - ***base+offset accesses memory once, Indirect mode accesses memory twice.***

# LD: Load Direct

- Assembler Inst.

  LD DR, LABEL ; DR <= Mem[LABEL]

- Encoding

  0010 DR PCoffset9 ; EA = (PC) + SEXT( PCoffset9 )
  = (PC) + SEXT( IR[8:0] )

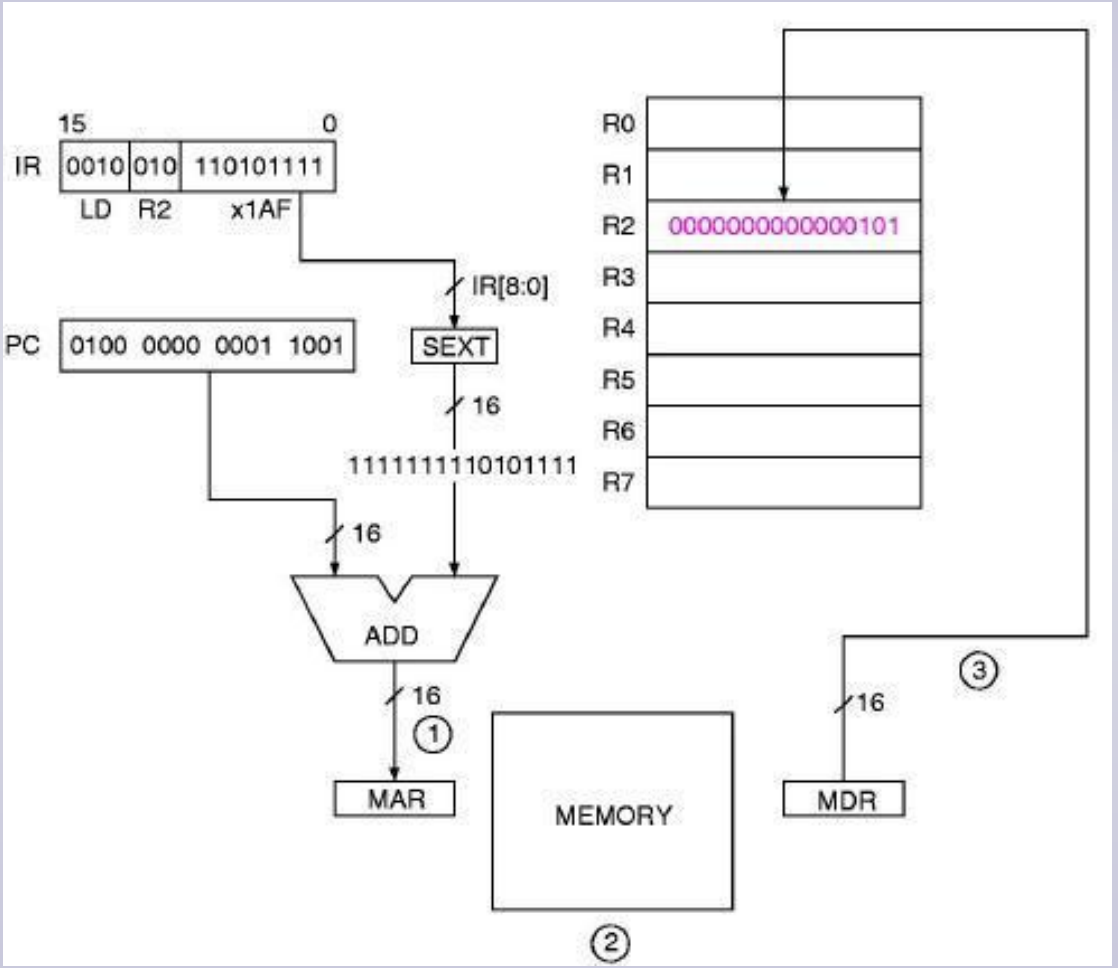
- Examples

  LD R2, param ; R2 <= Mem[param]

**Notes: The LABEL must be within +256/-255 lines of the instruction.**
**Condition codes are set.**

# LD data path

## LD R2, x1AF



# LDI: Load Indirect

- Assembler Inst.

  LDI DR, LABEL ; DR <= Mem[Mem[LABEL]]

- Encoding

  1010 DR PCoffset9 ; EA = Mem[ (PC) + SEXT( PCoffset9 ) ]
  = Mem[ (PC) + SEXT( IR[8:0] ) ]

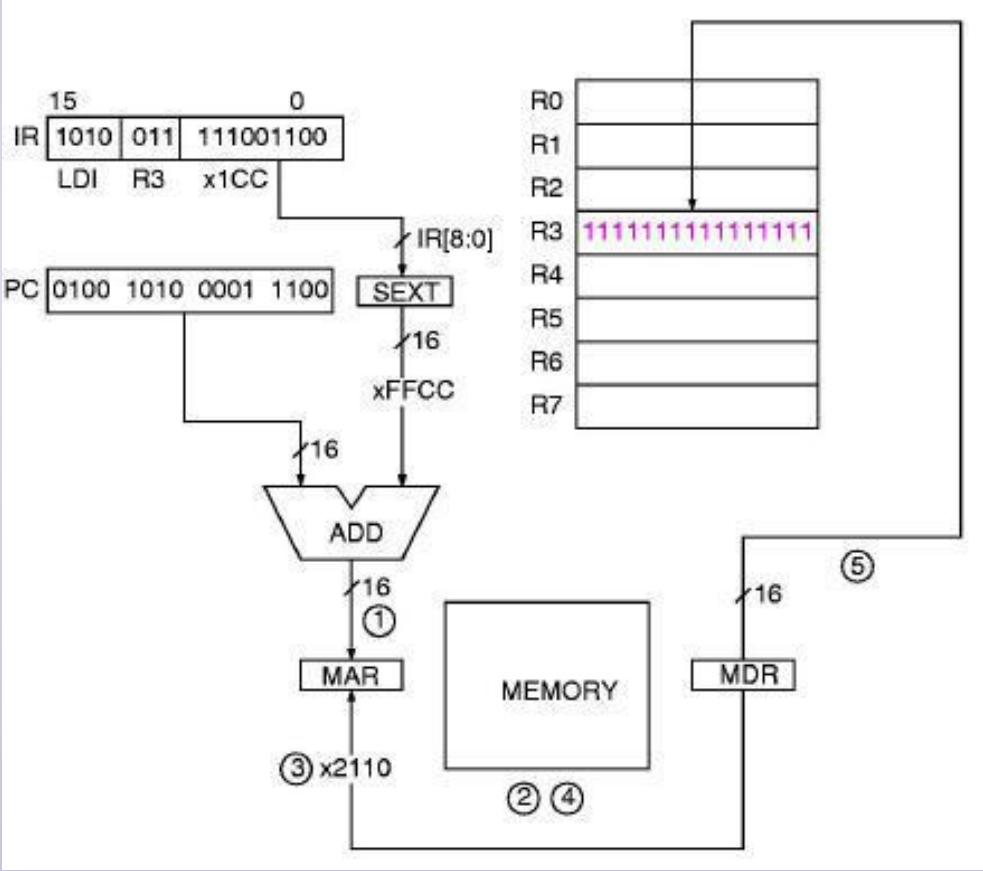
- Examples

  LDI R2, POINTER ; R2 <= Mem[Mem[POINTER]]

**Notes: The LABEL must be within +256/-255 lines of the instruction.**
**Condition codes are set.**

# LDI data path

## LDI R3, x1CC



# LDR: Load Base+Offset

- Assembler Inst.

  LDR DR, BaseR, offset ; DR <= Mem[ (BaseR) + SEXT( IR[5:0] ) ]

- Encoding

  0110 DR BaseR offset6

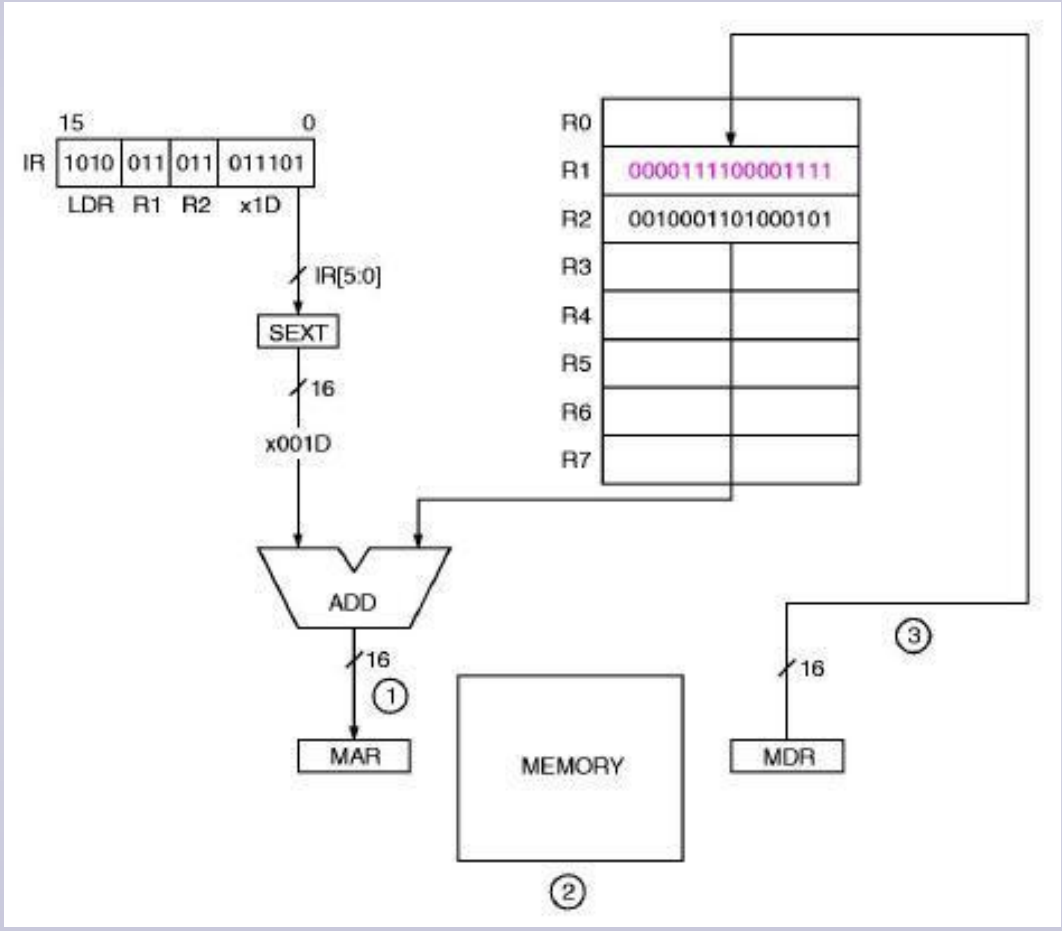
- Examples

  LD R2, R3, #15 ; R2 <= Mem[ (R3)+15 ]

**Notes: The 6 bit offset is a 2's complement number, so range is -32 to +31.**
**Condition codes are set.**

# LDR data path

## LDR R1, R2, x1D



# LEA: Load Effective Address

- Assembler Inst.

  LEA DR, LABEL ; DR <= LABEL

- Encoding

  1110 DR offset9 i.e. address of LABEL = (PC) + SEXT(offset9)

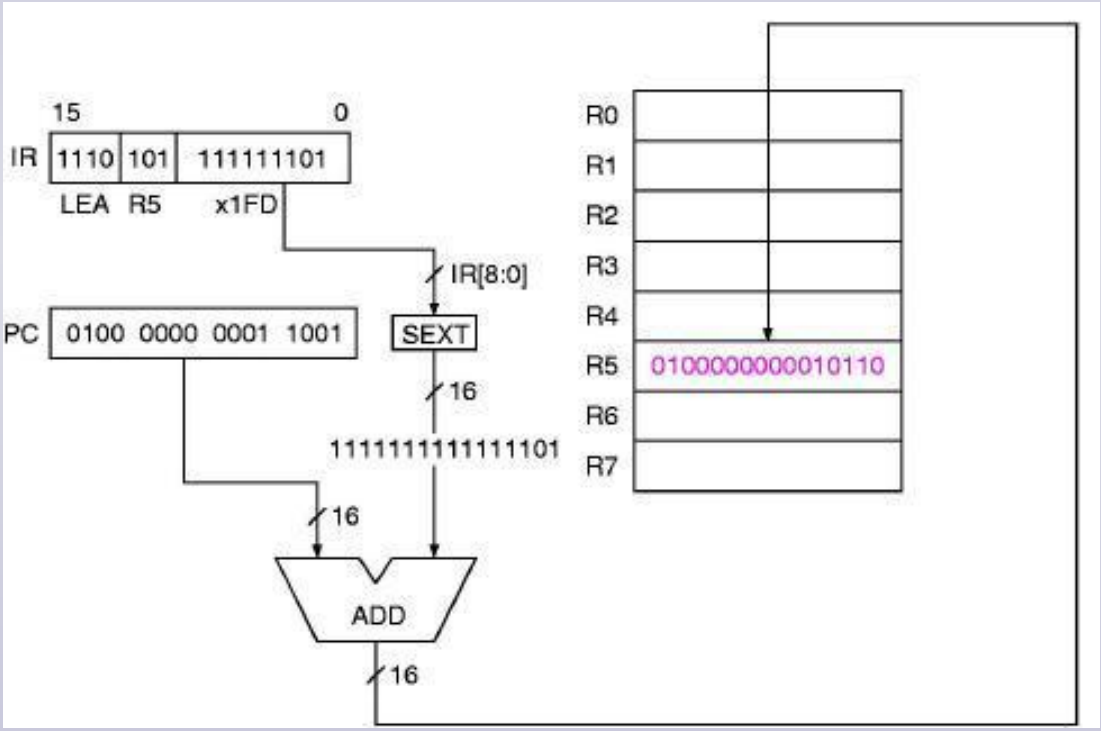
- Examples

  LEA R2, DATA ; R2 gets the address of DATA = (PC) + Sext( IR[8:0] )

**Notes: The LABEL must be within +/- 256 lines of the instruction.**
**Condition codes are set.**

# LEA data path

## LEA R5, # -3







Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

# ST: Store Direct

- Assembler Inst.

```
ST  SR, LABEL      ; [ LABEL ] <= (SR)
                   ; The value stored in SR is copied to the
                   ; memory address identified as LABEL
```

- Encoding

```
0011  SR  offset9
```

- Examples

```
ST   R2, VALUE     ; [ VALUE ] <= (R2)
```

**Notes: The LABEL must within +/- 256 lines of the instruction.**
**Condition codes are NOT set.**

# STI: Store Indirect

- Assembler Inst.

```
STI  SR, LABEL    ; [ Mem[LABEL] ] <= (SR)
                  ; The value stored in SR is copied to the
                  ; memory address pointed to by LABEL
```

- Encoding

```
0011  SR  offset9
```

- Examples

```
STI  R2, POINTER  ; [ Mem[POINTER] ] <= (R2)
```

**Notes: The LABEL must be within +/- 256 lines of the instruction.**
**Condition codes are NOT set.**

# STR: Store Base+Offset

- Assembler Inst.

```
STR SR, BaseR, offset6    ; [ (BaseR)+SEXT(offset6) ] <= (SR)
                          ; EA = (BaseR) + SEXT(offset6)
                          ; The contents of SR are copied to EA
```

- Encoding

```
0111  SR  BaseR offset6
```

- Examples

```
STR   R2, R4, #15   ;      [ (R4) +15 ] <= (R2)
```

**Notes: The offset is sign-extended to 16 bits.**
**Condition codes are not set.**

# Addressing Examples

## What is the EA for the following instructions?

**Given:**

**(PC) = x2081, (R6) = x2035, LOC = x2044, Mem[x2044] = x3456**

**LDI R2, LOC**
Encoding:
1010 010 1 1100 0011
Indirect addressing:
EA = Mem[?] = ?

**LDR R1, R6, #12**
Encoding:
0110 001 110 00 1100
Base+Offset addressing:
EA = ? = ?

**ADD R1, R3, R2**
Register addressing:
DR = R1, SR1 = R3, SR2 = R2
DR <= ?

**ADD R5, R1, #15**
Immediate addressing:
DR = R5, SR1 = R1, S2 = 15
DR <= ?

**LD R1, LOC**
Direct addressing:
DR <= ?

# Control Instructions

- **Change the Program Counter**
  - Conditionally or unconditionally
  - Store the original PC (subroutine calls) or not (go-to)
- **LC-3 Control Instructions**
  - BRx, JMP/RET, JSR/JSRR, TRAP, RTI
    - *BRx uses PC-Relative addressing with 9-bit offset*
    - *JSR uses PC-Relative addressing with 11-bit offset*
    - *JMP/RET & JSRR use base+offset addressing with zero offset*
    - *we'll deal with TRAP & RTI later*

# BR: Conditional Branch

- Assembler Inst.

  BRx LABEL ; PC <= LABEL iff condition evaluates to true

  where x = n, z, p, nz, np, zp, or nzp

  **Branch to LABEL iff the selected condition code are set**

- Encoding

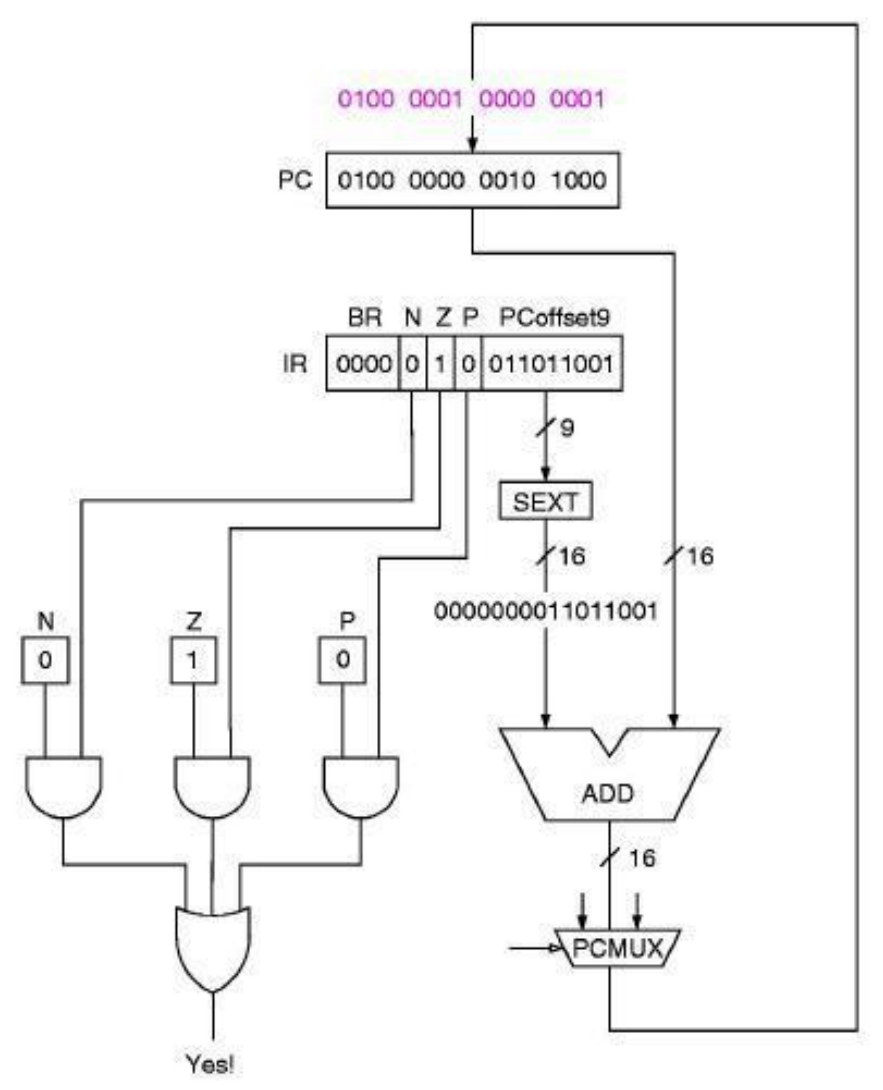
  0000 n z p PCoffset9 ; PC <= (PC) + Sext( IR[8:0] ) if condition

- Examples

  BRzp LOOP ; branch to LOOP if previous op returned zero or positive.

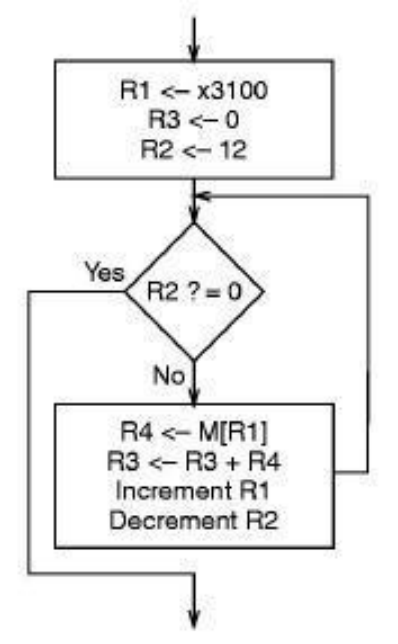
# BR data path

**BRz x0D9**

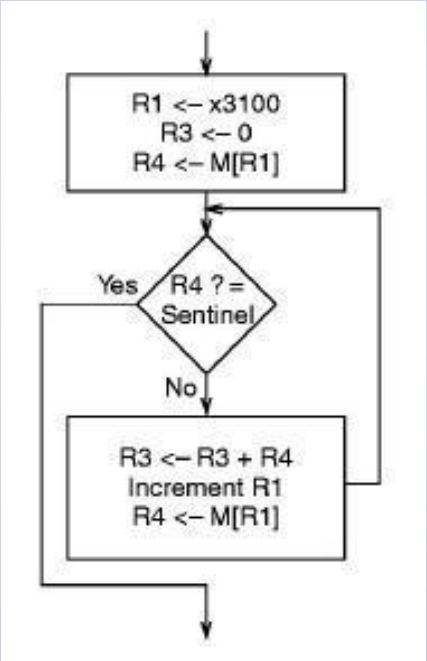


# Building loops using BR

## Counter control




## Sentinel control



# JMP: Jump or Go To

- Assembler Inst.

  JMP  BaseR  ; PC <= (BaseR)

  **Take the next instruction from the address stored in BaseR**

  ***(this is the same memory addressing mode as LDR & STR, but with no offset)***

- Encoding

  1100  000 BaseR 00 0000

- Example

  JMP R5  ; if (R5) = x3500, the address x3500 is written to the PC

# TRAP Instruction

- Used to invoke an operating system service.
- Trap vector table: a list of locations of the service call routines.
- TRAP has one operand, the trap vector:

  PC is set to the value stored at that location of the vector table.
- Some special trap vectors:

* x20: input a character from the keyboard
* x23: input a character from the keyboard, with prompt & echo
* x21: output a character to the console display
* x25: halt the program

- More details later

# TRAP: Invoke a system routine

- Assembler Inst.

  TRAP trapvec ; step 1: R7 <= (PC)

  ; step 2: PC <= Mem[ Zext( IR[7:0] ) ]

  *Note we ZERO extend the trapvec, not SIGN extend*

- Encoding

  1111 0000 trapvect8

- Example

  TRAP x22 ; invoke the BIOS routine PUTS

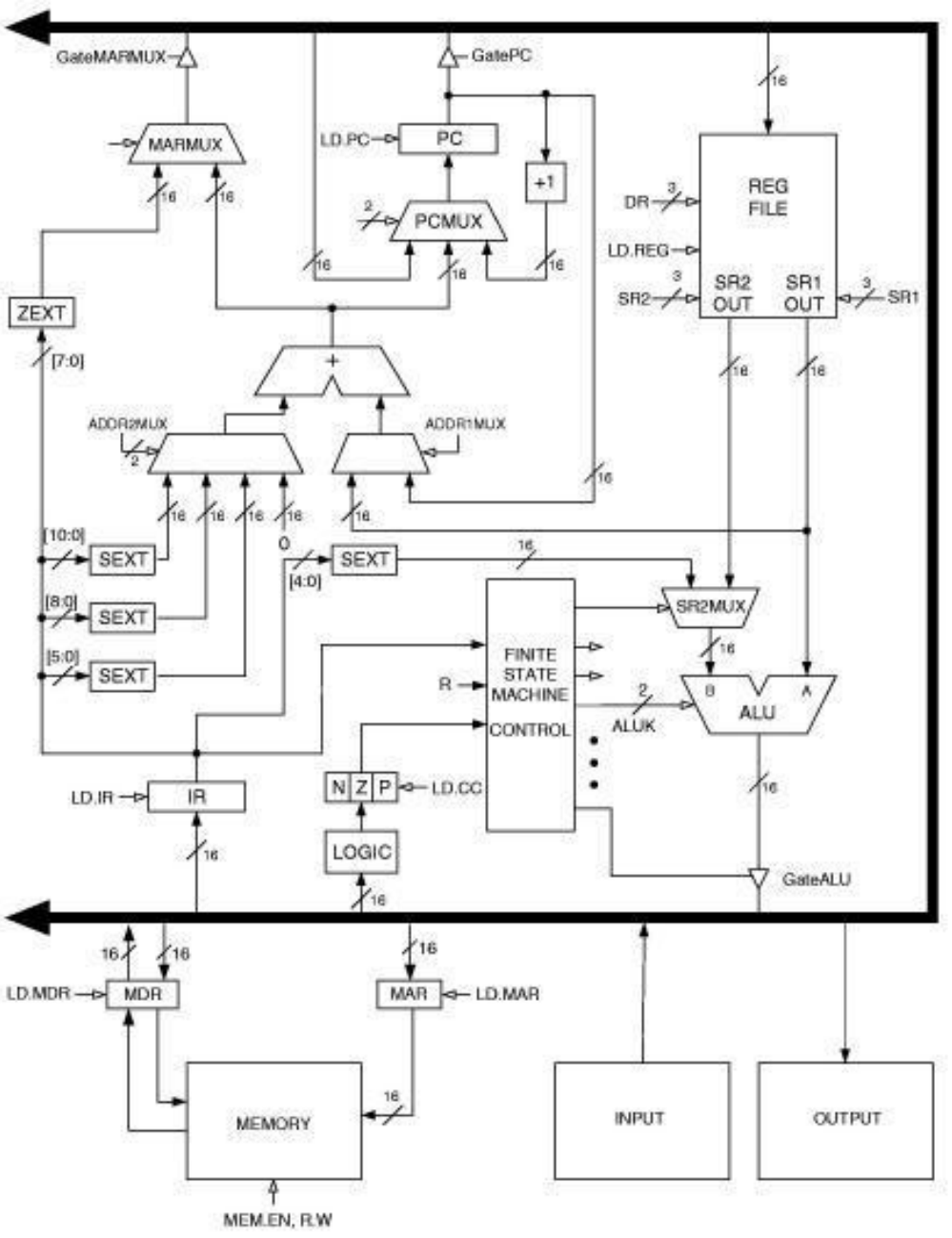
Meaning: R7 <= (PC) *(for eventual return from subroutine)*

PC <= Mem[ x0022 ]

*i.e. PC gets the address stored at entry trapvect8 in the trap vector table.*

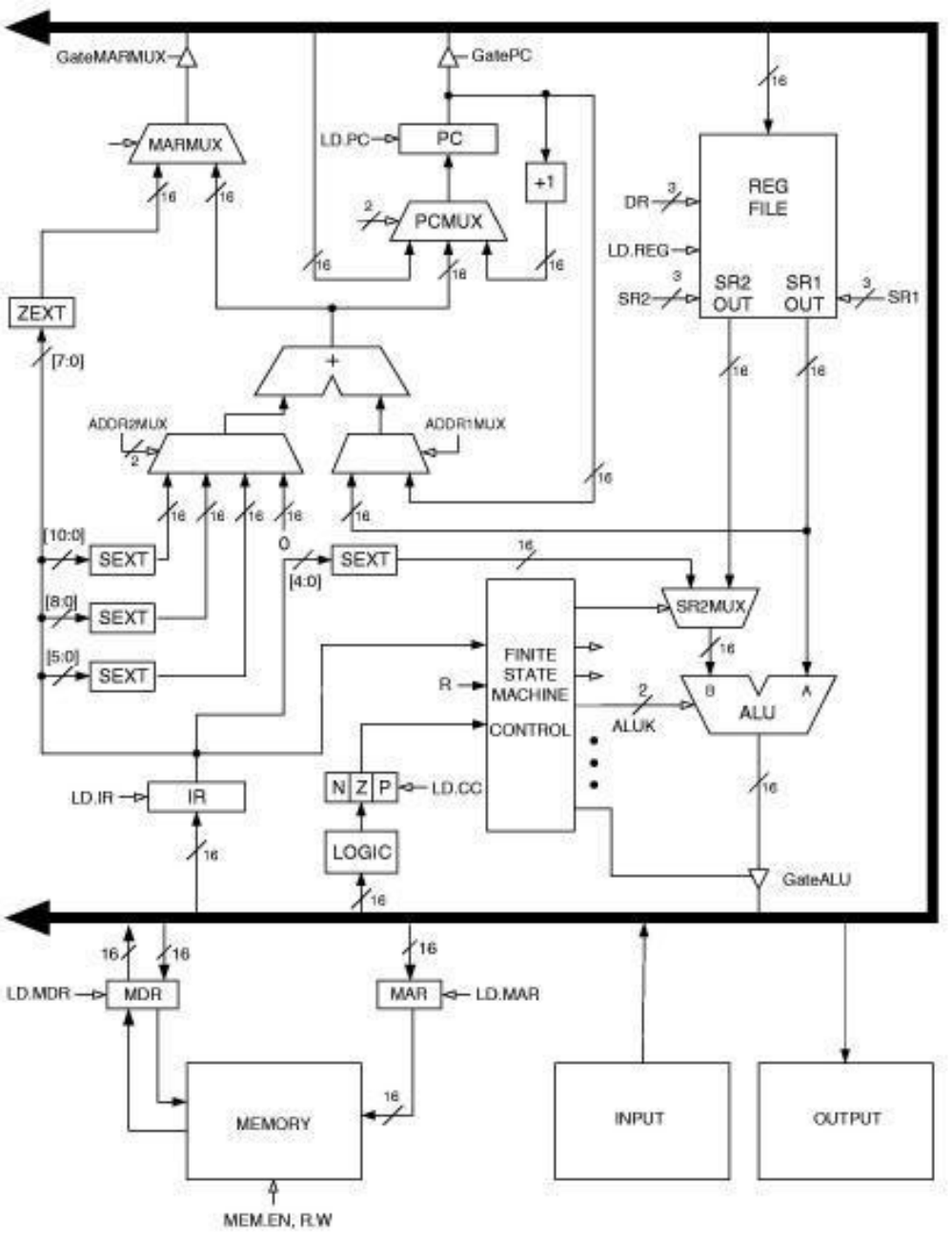
# Data Path - 1

- **Global Bus**
  - *16-bit, data & address*
  - *connects all components*
  - *is shared by all*
- **Memory**
  - *Memory Address Register: MAR*
    - address of location to be accessed
  - *Memory Data Register: MDR*
    - data loaded or to be stored



# Data Path - 2

- **ALU & Registers**
  - *Two ALU sources*
    - source 1: register
    - source 2: register or IR
  - *Result: goes onto bus, then to DR*
- **PC & PCMUX**
  - *PC sends address to MAR for instruction fetch*
  - *PCMUX: a 3:1 mux that selects the new PC*
    - Incremented PC
    - offset PC (9 or 11 bits)
    - offset BaseR (6 bits or 0)
    - TRAP vector contents



# Data Path - 3

- **MARMUX**
  - ***A 2:1 mux that selects the source of MAR***
    - **PC-Relative addressing**
    - **BaseR + offset addressing**
    - **Trap vector**