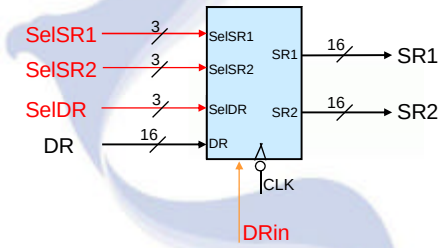
Term Project: Single Bus Processor

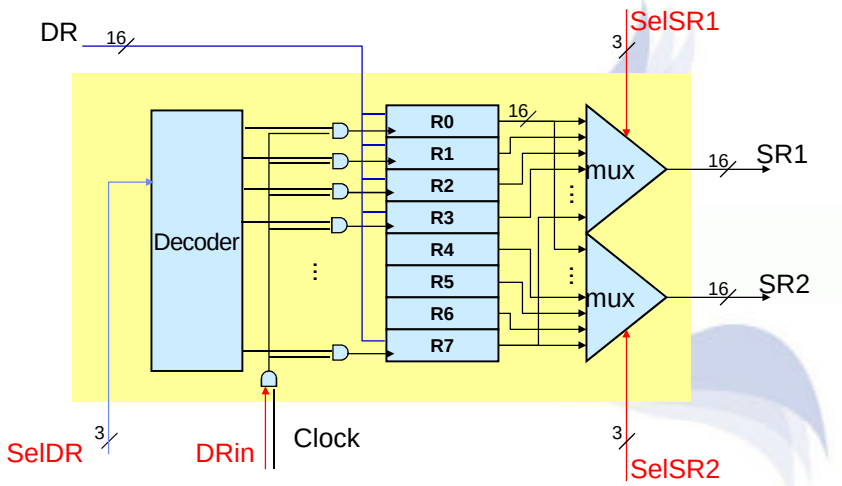
Design and simulate the LC-3 processor using VHDL

Register Files

- Register File consists of 8 registers:
  - Two 16-bit output busses: SR1 and SR2
  - One 16-bit input bus: DR
- Register is selected by:
  - SelSR1 selects the register to put on SR1
  - SelSR2 selects the register to put on SR2
  - SelDR selects the register to be written via DR when DRin is 1
- Behaves as a combinational logic block:
  - SelSR1 or SelSR2 valid => SR1 or SR2 valid after "access time."

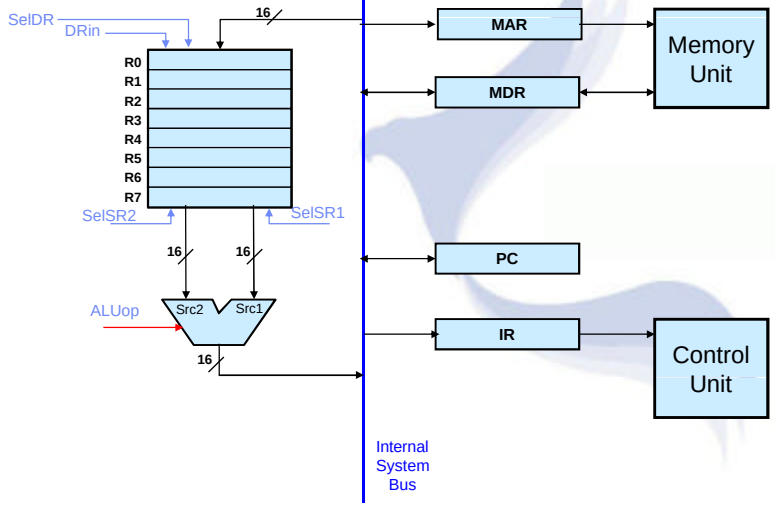


Register Files



Single Bus Architecture

- Accumulator based architecture



Instruction Set Architecture

- ISA = All of the programmer-visible components and operations of the computer
  - memory organization
    - address space -- how may locations can be addressed?
    - addressibility -- how many bits per location?
  - register set
    - how many? what size? how are they used?
  - instruction set
    - opcodes
    - data types
    - addressing modes
- ISA provides all information needed for someone that wants to write a program in machine language (or translate from a high-level language to machine language).

Memory and Registers

- Memory
  - address space: 2^{16} locations (16-bit addresses)
  - addressability: 16 bits
- Registers
  - temporary storage, accessed in a single machine cycle
    - accessing memory generally takes longer than a single cycle
  - eight general-purpose registers: R0 - R7
    - each 16 bits wide
    - R7 is used to store the return address in subroutine jump
    - how many bits to uniquely identify a register?
  - other registers
    - not directly addressable, but used by (and affected by) instructions
    - PC (program counter), CC (condition codes), MAR (memory address register), MDR (memory data register)

Instruction Set

- Opcodes
  - 15 opcodes
  - **Operate** instructions: ADD, AND, NOT
  - **Data movement** instructions: LD, LDI, LDR, LEA, ST, STR, STI
  - **Control** instructions: BR, JSR, JMP, RET, TRAP
  - some opcodes set/clear *condition codes*, based on result:
    - N = negative, Z = zero, C = carry
- Data Types
  - 16-bit 2's complement integer
- Addressing Modes
  - How is the location of an operand specified?
  - non-memory addresses: immediate, register
  - memory addresses: PC-relative, indirect, base+offset

Data Processing Instructions

- Only three operations: ADD, AND, NOT
- Source and destination operands are registers
  - These instructions *do not* reference memory.
  - ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.

Data Movement Instructions

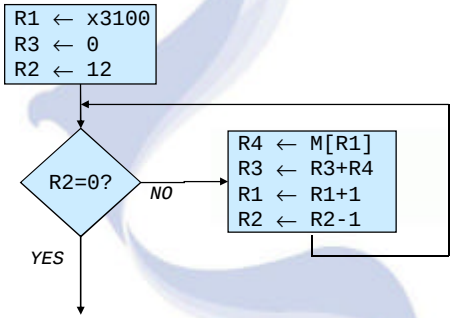
- Load -- read data from memory to register
  - LD: PC-relative mode
  - LDR: base+offset mode
  - LDI: indirect mode
- Store -- write data from register to memory
  - ST: PC-relative mode
  - STR: base+offset mode
  - STI: indirect mode
- Load effective address -- compute address, save in register
  - LEA: immediate mode
  - *does not access memory*

Control Instructions

- Used to alter the sequence of instructions (by changing the Program Counter)
- Conditional Branch : BR[nzc]
  - branch is *taken* if a specified condition is true
    - signed offset is added to PC to yield new PC
  - else, the branch is *not taken*
    - PC is not changed, points to the next sequential instruction
- Unconditional Branch : JMP, JSR, RET
  - always changes the PC
- TRAP
  - changes PC to the address of an OS "service routine"
  - routine will return control to the next instruction (after TRAP)

Branch Instruction

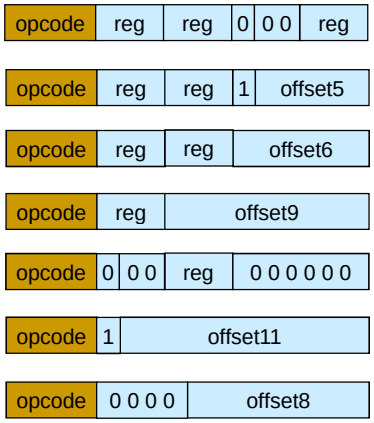
- Branch specifies one or more condition codes.
- If the set bit is specified, the branch is taken.
  - PC-relative addressing: target address is made by adding signed offset (IR[8:0]) to current PC.
  - Note: PC has already been incremented by FETCH stage.
  - Note: Target must be within 256 words of BR instruction.
- If the branch is not taken, the next sequential instruction is executed.



Condition Codes

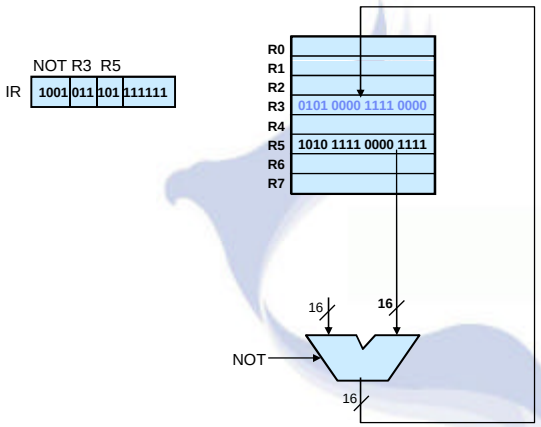
- LC-3 has three condition code registers:
  N -- negative
  Z -- zero
  C -- carry
- Set by any instruction that writes a value to a register (ADD, AND, NOT, LD, LDR, LDI, LEA)
- Exactly one will be set at all times
  - Based on the last instruction that altered a register

Instruction Format



Register Addressing Modes

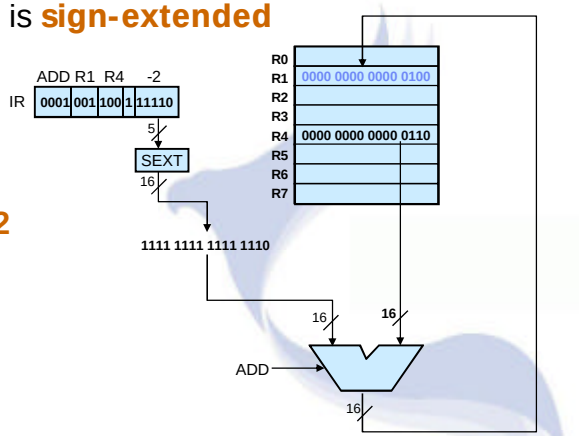
- **NOT R3, R5**



Immediate Addressing Mode

- Immediate field is **sign-extended**

ADD R1, R4, # -2

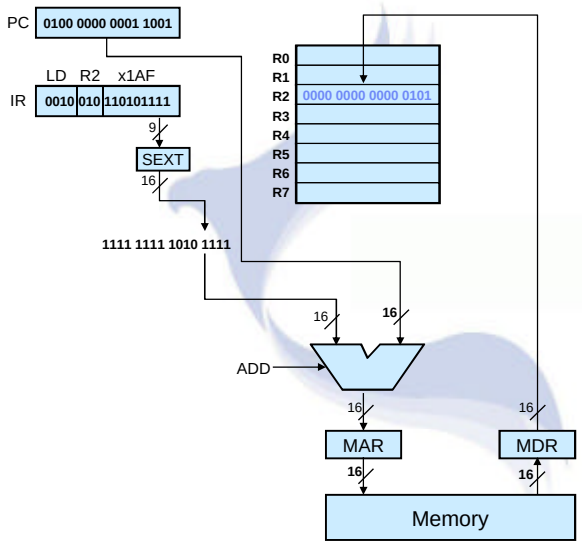


PC-Relative Addressing Mode

- Want to specify address directly in the instruction
  - But an address is 16 bits, and so is an instruction!
  - After subtracting 4 bits for opcode and 3 bits for register, we have 9 bits available for address.
- Solution:
  - Use the 9 bits as a *signed offset* from the current PC.
- 9 bits: $-256 \le \text{offset} \le +255$
- Can form any address X, such that: $PC-256 \le X \le PC+255$
- Remember that PC is incremented as part of the FETCH phase;
- This is done before the EVALUATE ADDRESS stage.

PC-Relative Addressing Mode

- LD R2, x1AF

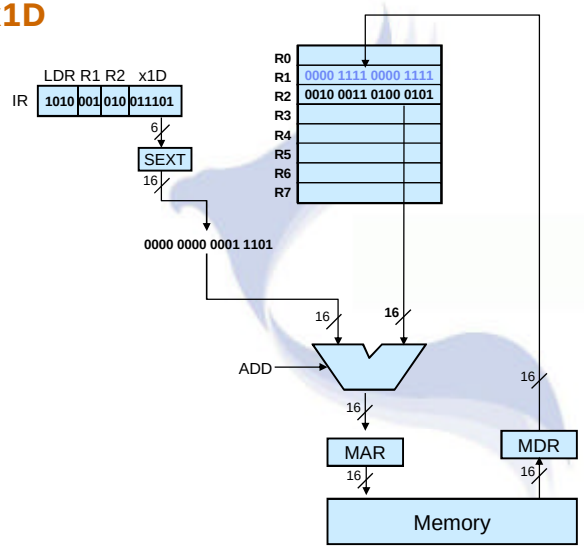


Base + Offset Addressing Mode

- With PC-relative mode, can only address data within 256 words of the instruction.
  - What about the rest of memory?
- Solution # 2:
  - Use a register to generate a full 16-bit address.
- 4 bits for opcode, 3 for src/dest register,3 bits for base register -- remaining 6 bits are used as a signed offset.
  - **Offset is sign-extended before adding to base register.**

Base + Offset Addressing Mode

- LDR R1, R2, x1D

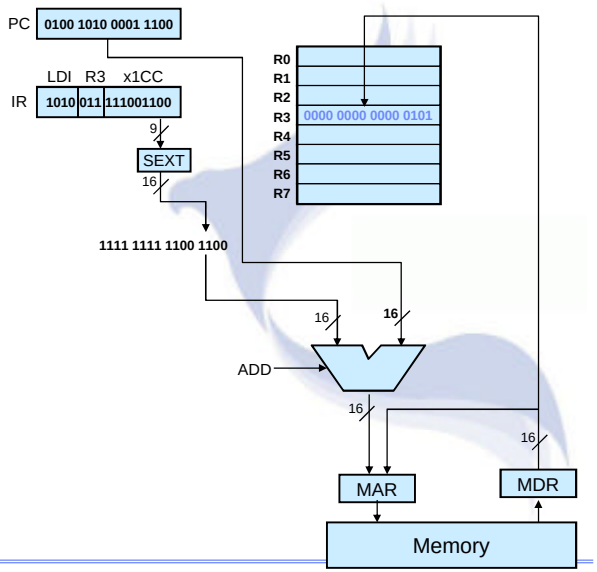


Indirect Addressing Mode

- With PC-relative mode, can only address data within 256 words of the instruction.
  - What about the rest of memory?
- Solution # 1:
  - Read address from memory location, then load/store to that address.
- First address is generated from PC and IR (just like PC-relative addressing), then content of that address is used as target for load/store.

Indirect Addressing Mode

LDI R3, x1CC



Instruction Set

BR	0 0 0 0	N	Z	C	PCoffset9											PC-Relative
ADD	0 0 0 1	DR			SR1	0	0 0	SR2								Register
	0 0 0 1	DR			SR1	1	Imm5									Immediate
LD	0 0 1 0	DR			PCoffset9											PC-Relative
ST	0 0 1 1	Rs			PCoffset9											PC-Relative
JSR	0 1 0 0	0	0 0		Base	0 0 0 0 0 0										Base register
	0 1 0 0	1	PCoffset11													PC-Relative
AND	0 1 0 1	DR			SR1	0	0 0	SR2								Register
	0 1 0 1	DR			SR1	1	Imm5									Immediate
LDR	0 1 1 0	DR			Base	offset6										Base+offset
STR	0 1 1 1	Rs			Base	offset6										Base+offset
NOT	1 0 0 1	DR			Rs	1 1 1 1 1 1										Register
LDI	1 0 1 0	DR			PCoffset9											Indirect
STI	1 0 1 1	Rs			PCoffset9											Indirect
JMP	1 1 0 0	0 0 0			Base	0 0 0 0 0 0										Base register
RET	1 1 0 1	0 0 0			R7	0 0 0 0 0 0										register
LEA	1 1 1 0	DR			PCoffset9											PC-Relative
TRAP	1 1 1 1	0 0 0 0			TrapVector8											Psudo-Direct

Instruction Set

Operation	opcode	Example	Macro-operation	Addressing mode
Data Processing	1001	NOT R1, R2	R1← ~R2	Register
	0101	AND R1, R2, R3	R1← R2 & R3	Register
		AND R1, R2, #x2	R1← R2 & x2	Immeadate
	0001	ADD R1, R2, R3	R1← R2 + R3	Register
		ADD R1, R2, #x2	R1← R2 + x0002	Immeadate
Data Movement	0010	LD R1, x123	R1← M[PC+x0123]	Relative
	0110	LDR R1, R2, x12	R1← M[R2+x12]	Base-Offset
	1010	LDI R1, x123	R1← M[M[PC+x123]]	Indirect
	1110	LEA R1, x123	R1← PC + x123	Relative
	0011	ST R1, x123	M[PC+x0123] ← R1	Relative
	0111	STR R1, R2, x12	M[R2+x12] ← R1	Base-Offset
	1011	STI R1, x123	M[M[PC+x123]] ← R1	Indirect
Program Control	0000	BR[nzc] x123	If [nzc] PC← PC + x123	Relative
	1100	JMP R1	PC← R1	Register
	0100	JSR R1	R7←PC; PC←R1	Register
		JSR x123	R7←PC; PC← PC + x123	Relative
	1101	RET	PC← R7	Implicit

Sample Program

Address	Instruction	Comments
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	R1 ¬ x3100 (PC+ 0xFF)
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	R3 ¬ 0
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ¬ 0
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	R2 ¬ 12
x3004	0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1	If Z, goto x300A (PC+ 5)
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	Load next value to R4
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1	Add to R3
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	Increment R1 (pointer)
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	Decrement R2 (counter)
x3009	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0	Goto x3004 (PC-6)

Program (1 of 2)

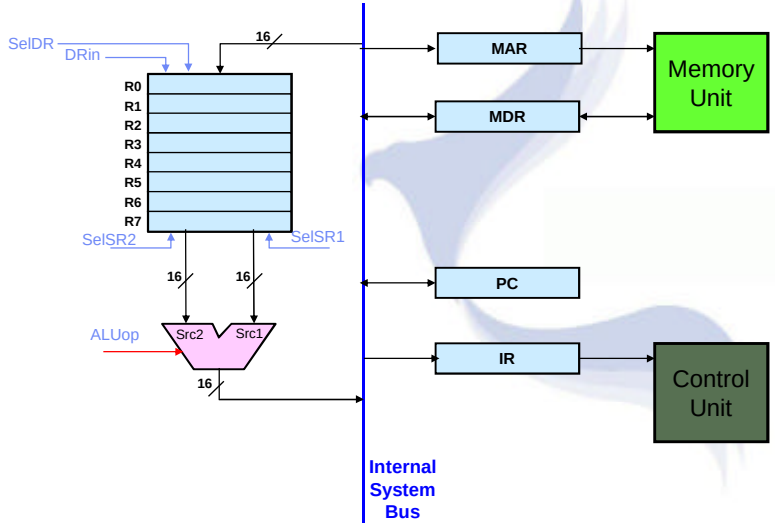
Address	Instruction	Comments
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ¬ 0 (counter)
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	R3 ¬ M[x3102] (ptr)
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	Input to R0 (TRAP x23)
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	R1 ¬ M[R3]
x3004	0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0	R4 ¬ R1 – 4 (EOT)
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	If Z, goto x300E
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	R1 ¬ NOT R1
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	R1 ¬ R1 + 1
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	R1 ¬ R1 + R0
x3009	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1	If N or C, goto x300B

Program (2 of 2)

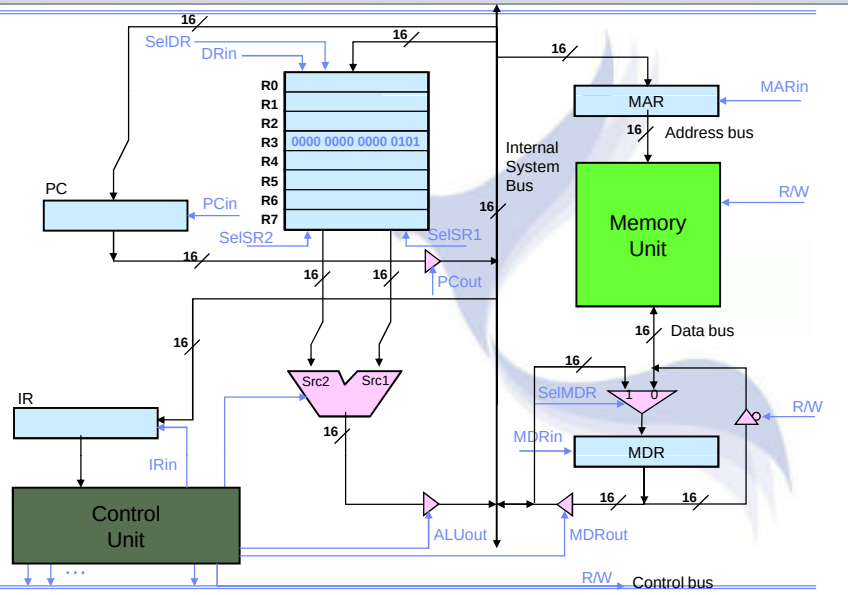
Address	Instruction	Comments
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	R2 ¬ R2 + 1
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	R3 ¬ R3 + 1
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	R1 ¬ M[R3]
x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	Goto x3004
x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0	R0 ¬ M[x3013]
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	R0 ¬ R0 + R2
x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	Print R0 (TRAP x21)
x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	HALT (TRAP x25)
X3012	Starting Address of File	
x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	ASCII x30 ('0')

Single Bus Architecture

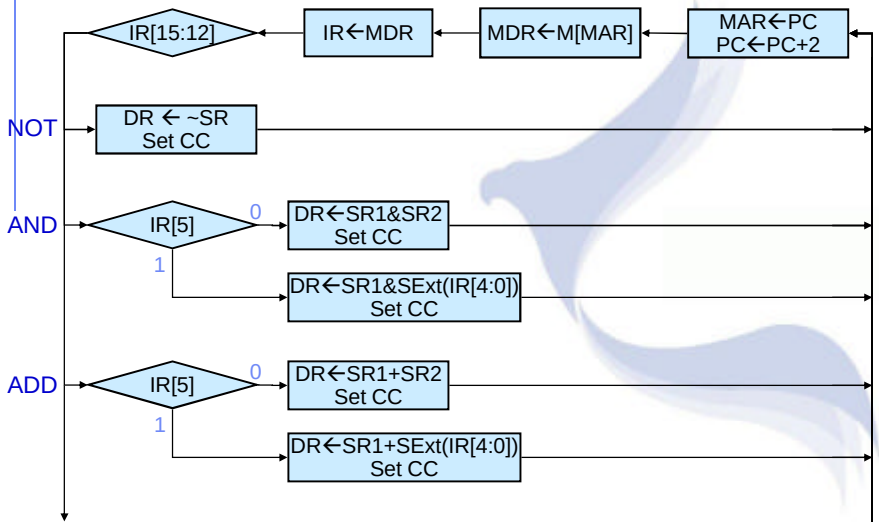
- Accumulator based architecture



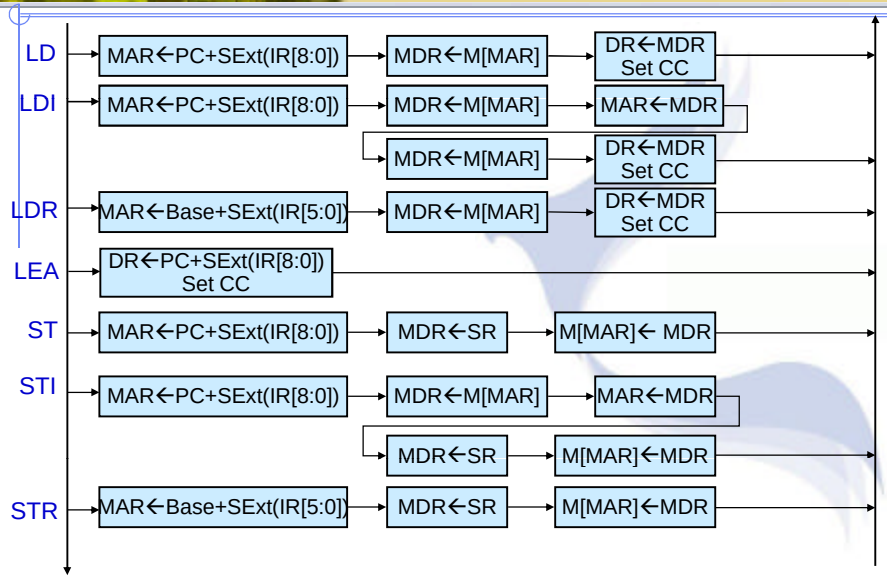
Basic Single Bus Architecture



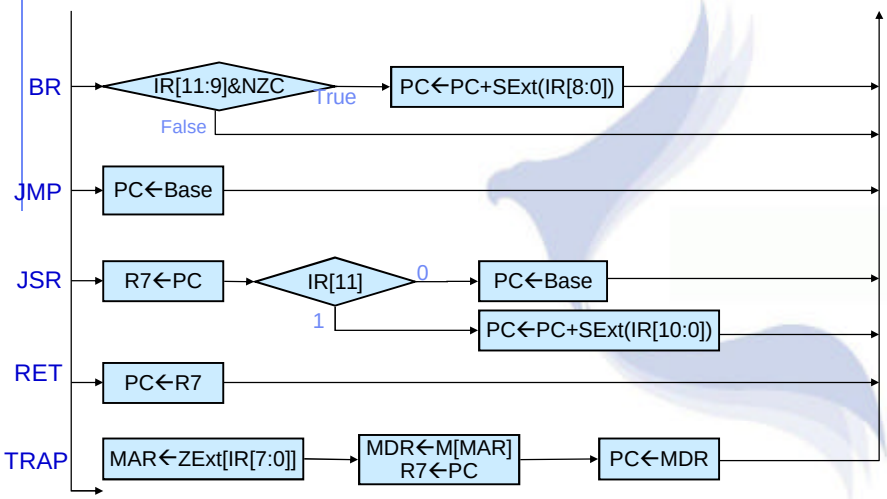
Register Transfer Logic



Register Transfer Logic

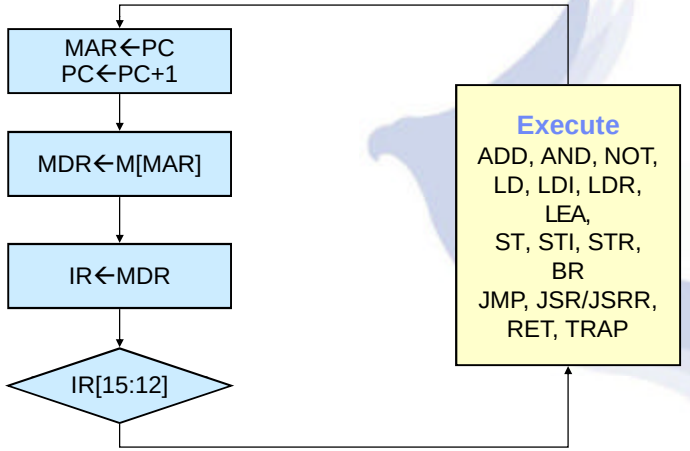


Register Transfer Logic

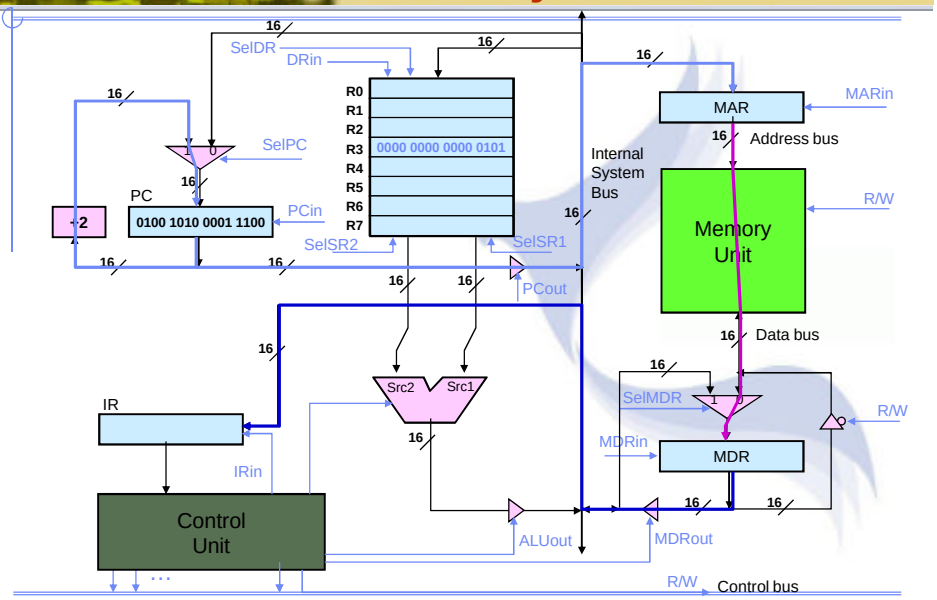


Instruction Cycle

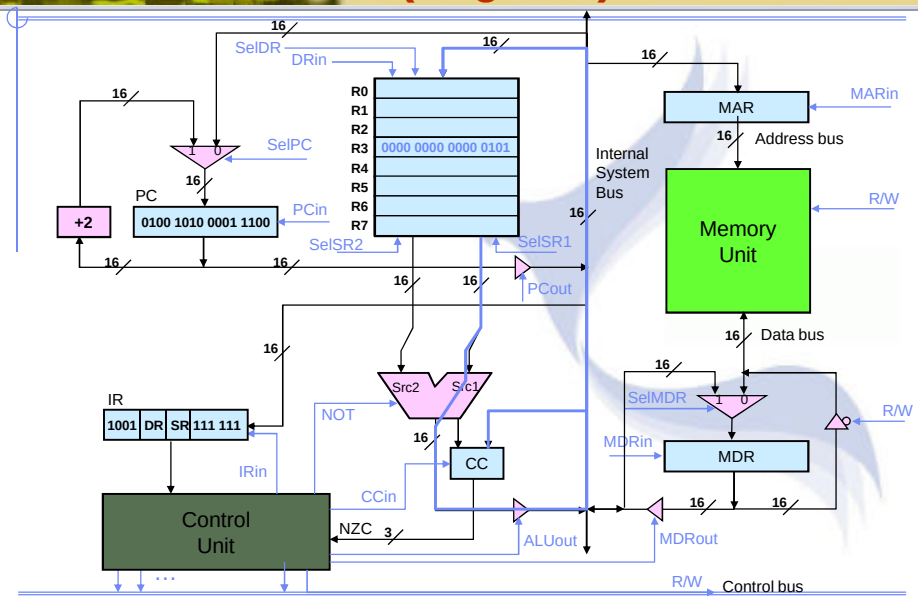
- Fetch cycle
- Execute cycle



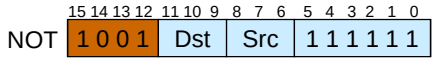
Fetch Cycle Architecture

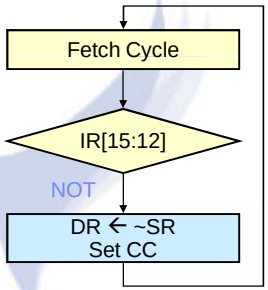


NOT (Register) Architecture



NOT (Register)

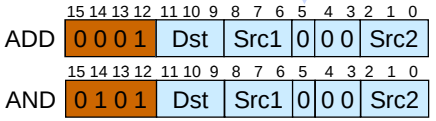


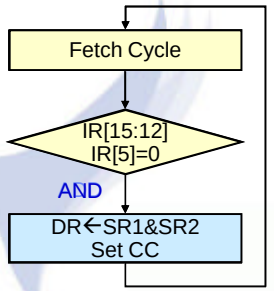


Note: Src and Dst could be the same register.

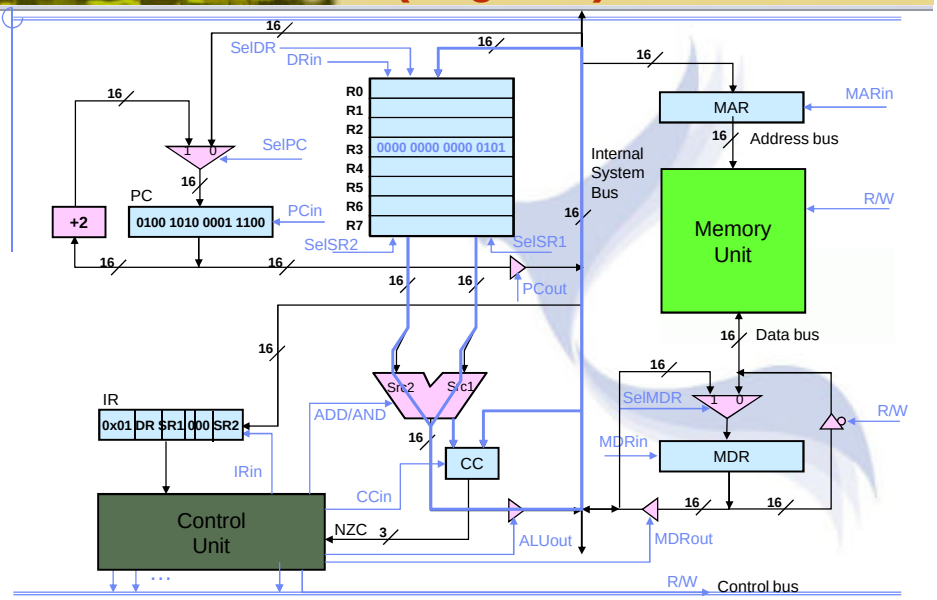
ADD/ AND (Register)

this zero means "register mode"

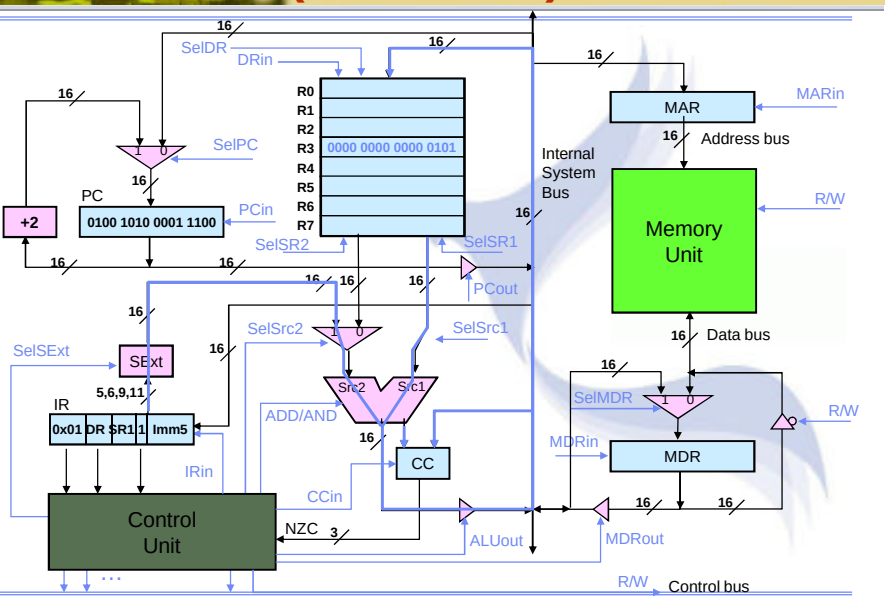




ADD/ AND (Register) Architecture

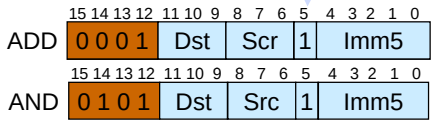


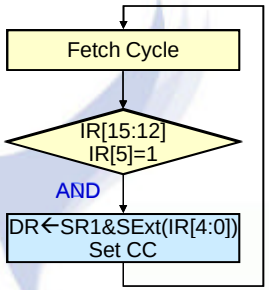
ADD/ AND (Immediate) Architecture



ADD/ AND (Immediate)

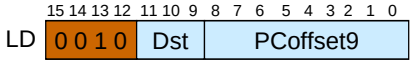
this zero means "immediate mode"

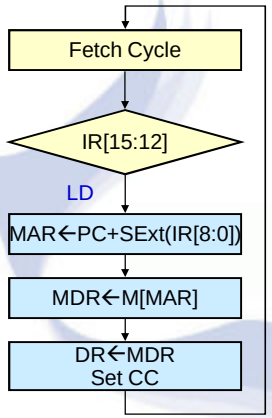




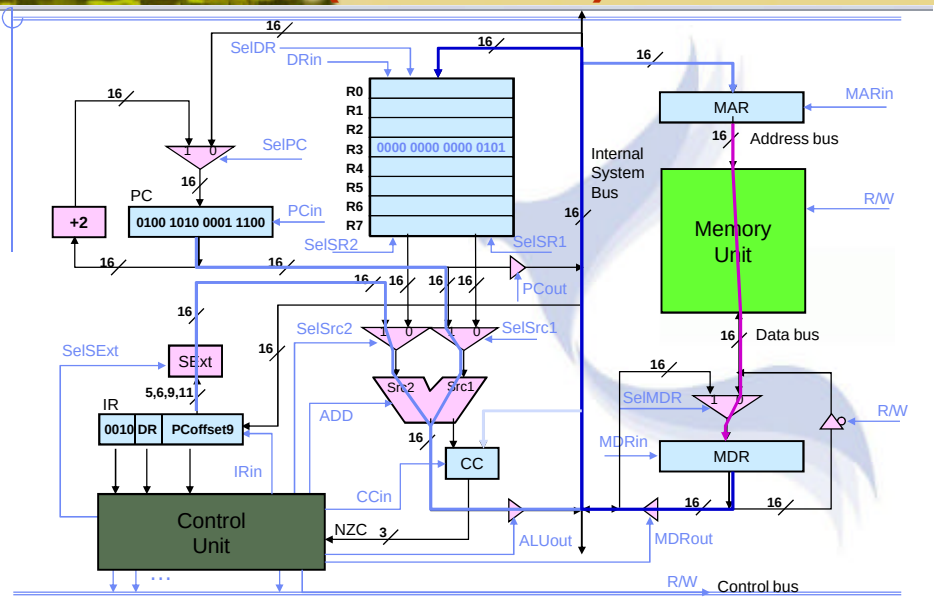
Note: Immediate field is ***sign-extended****.*

LD (PC-Relative)

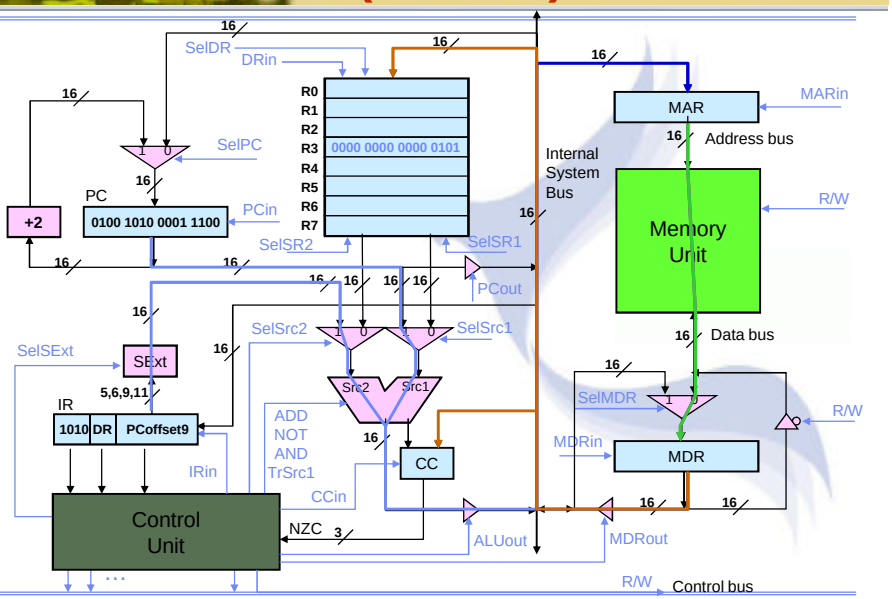




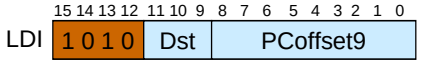
LD (PC-Relative) Architecture

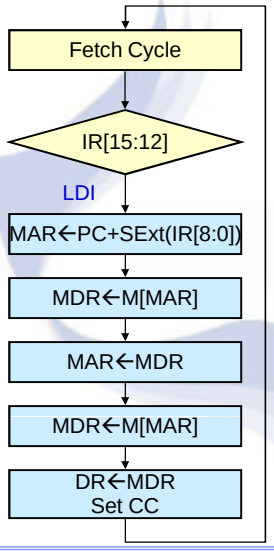


LDI (Indirect) Architecture

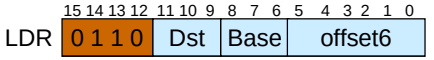


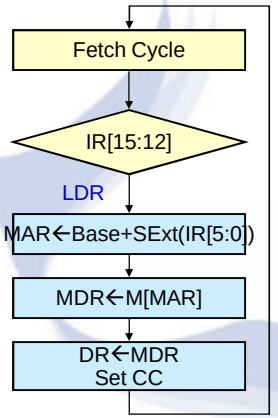
LDI (Indirect)



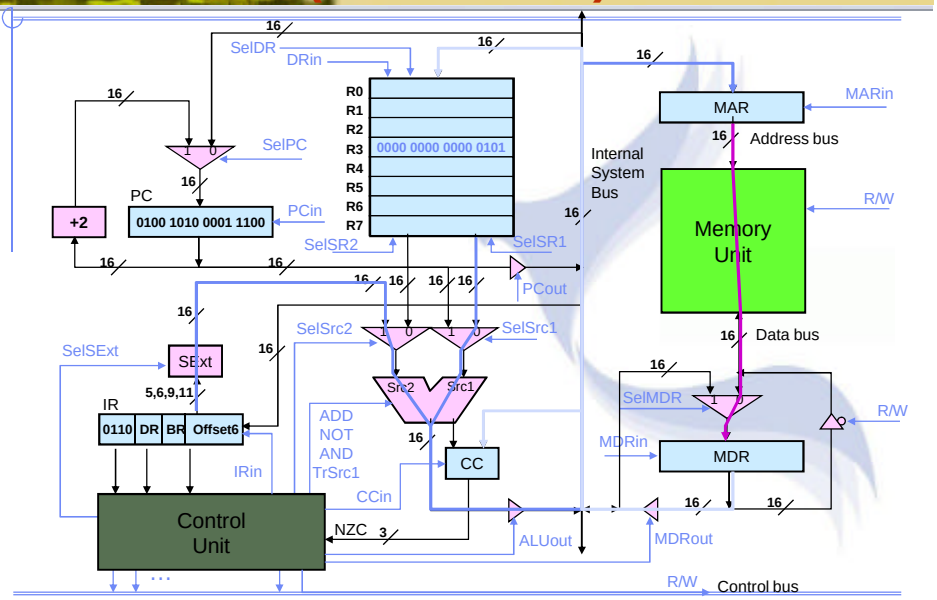


LDR (Base+ Offset)

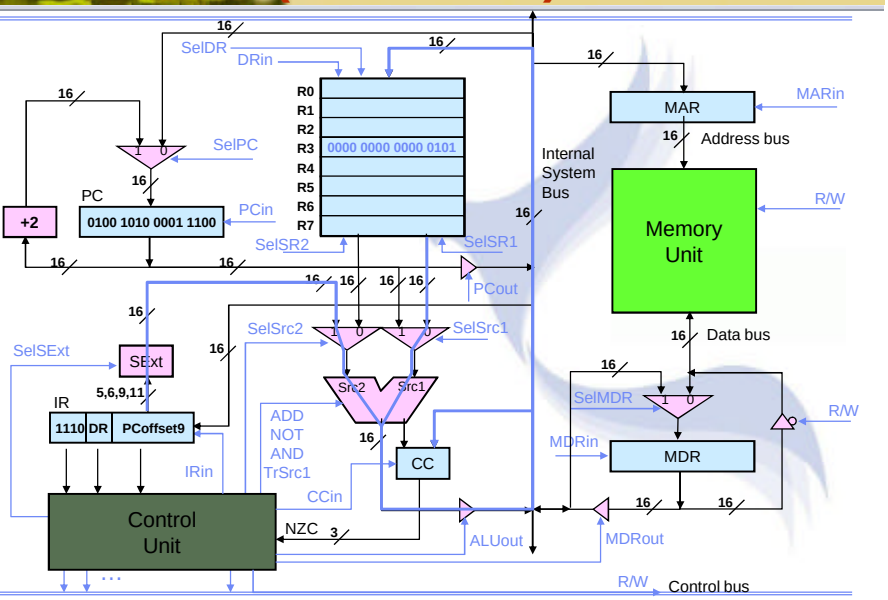




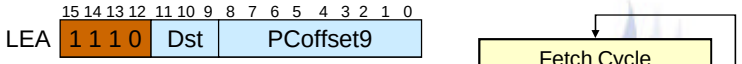
LDR (Base+ Offset) Architecture

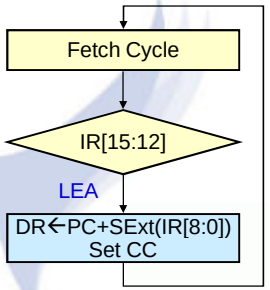


LEA (PC-Relative) Architecture

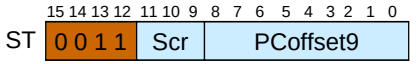


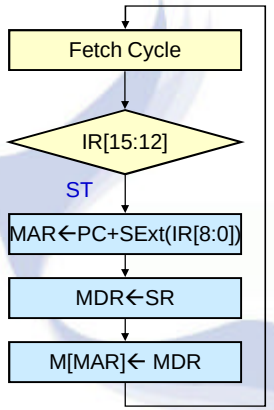
LEA (PC-Relative)



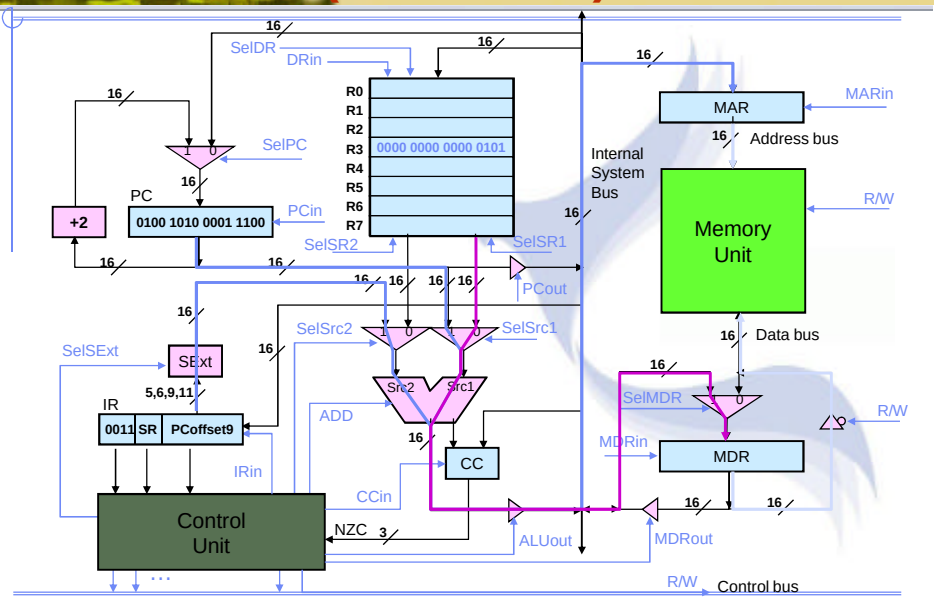


ST (PC-Relative)

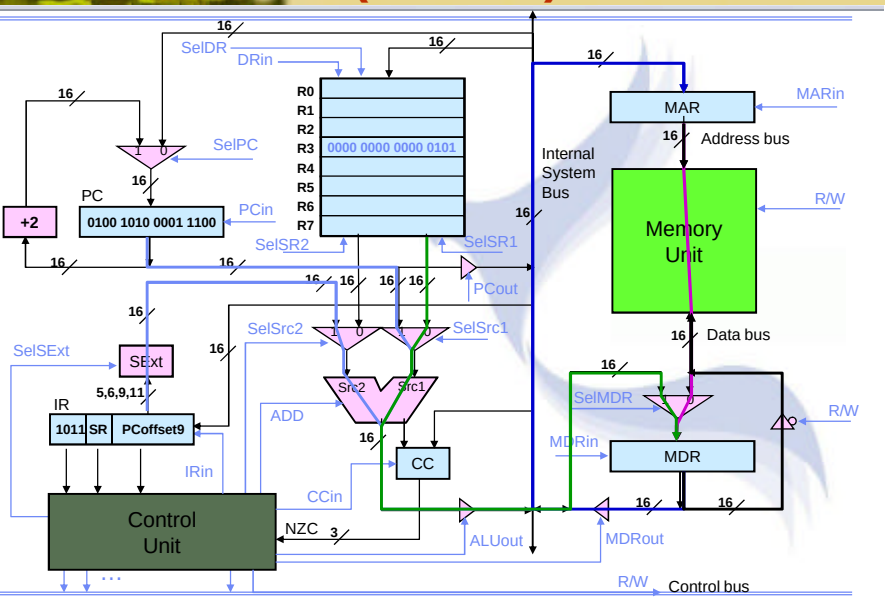




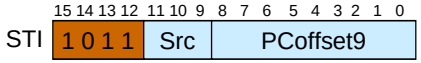
ST (PC-Relative) Architecture

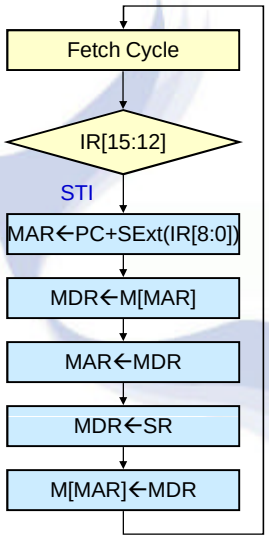


STI (Indirect) Architecture

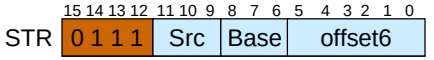


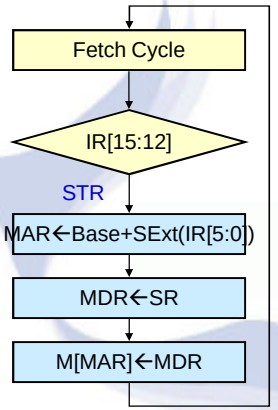
STI (Indirect)



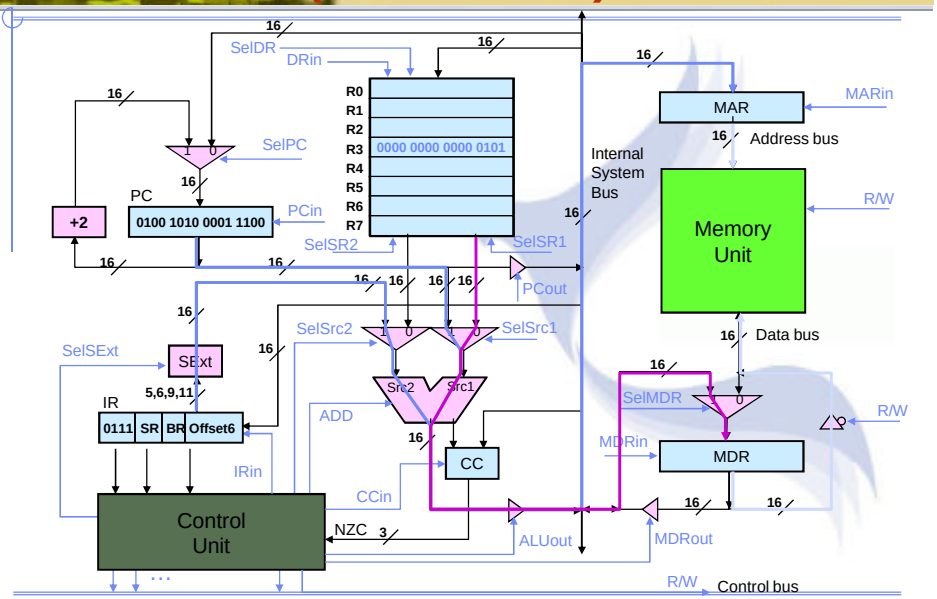


STR (Base+ Offset)

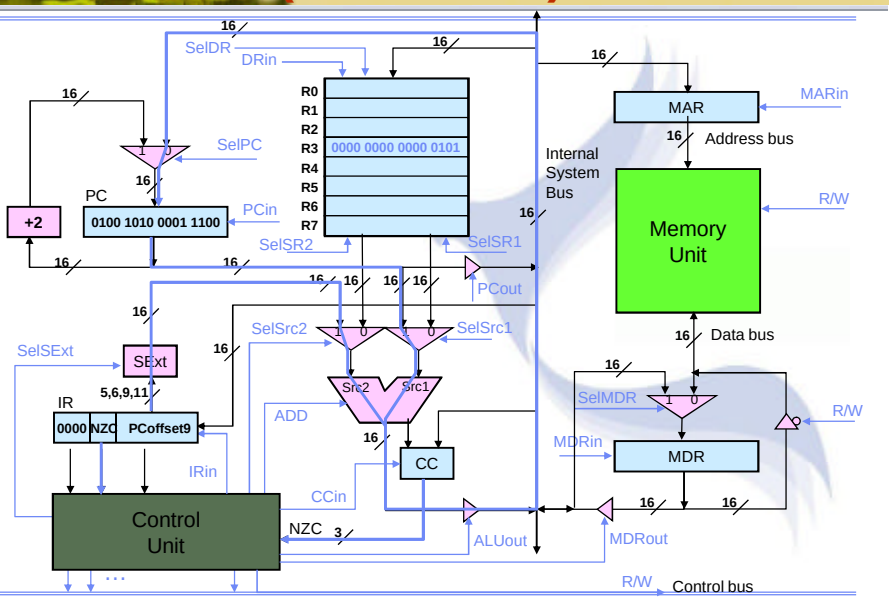




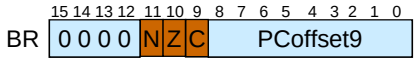
STR (Base+ Offset) Architecture

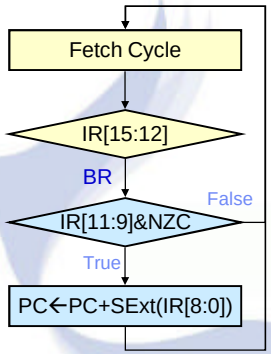


BR (PC-Relative) Architecture



BR (PC-Relative)

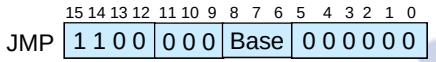


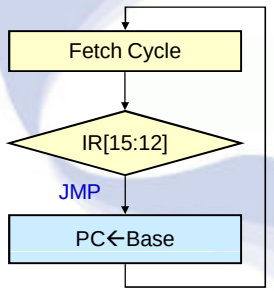


What happens if bits [11:9] are all zero? All one?

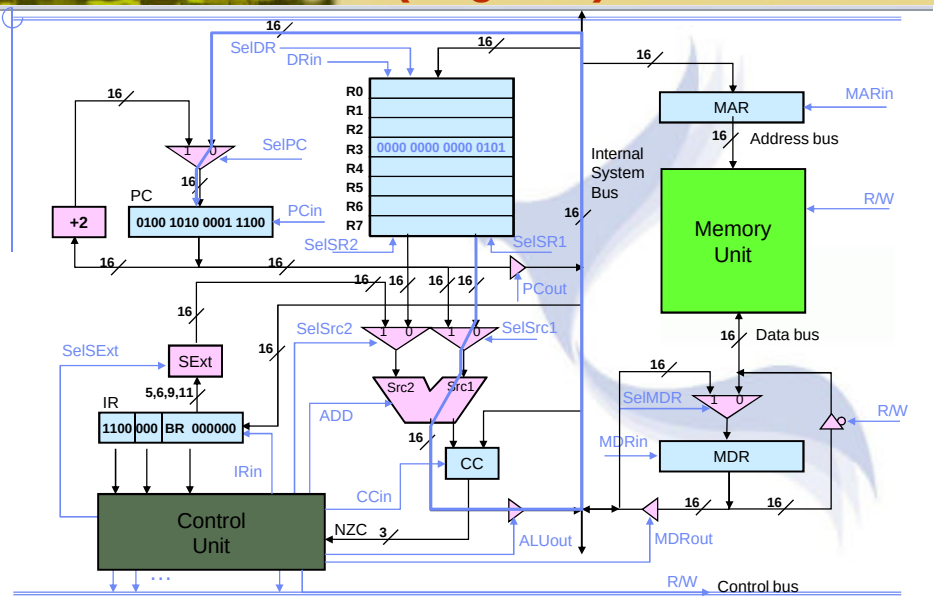
JMP (Register)

- Jump is an unconditional branch -- *always* taken.
  - Target address is the contents of a register.
  - Allows any target address.

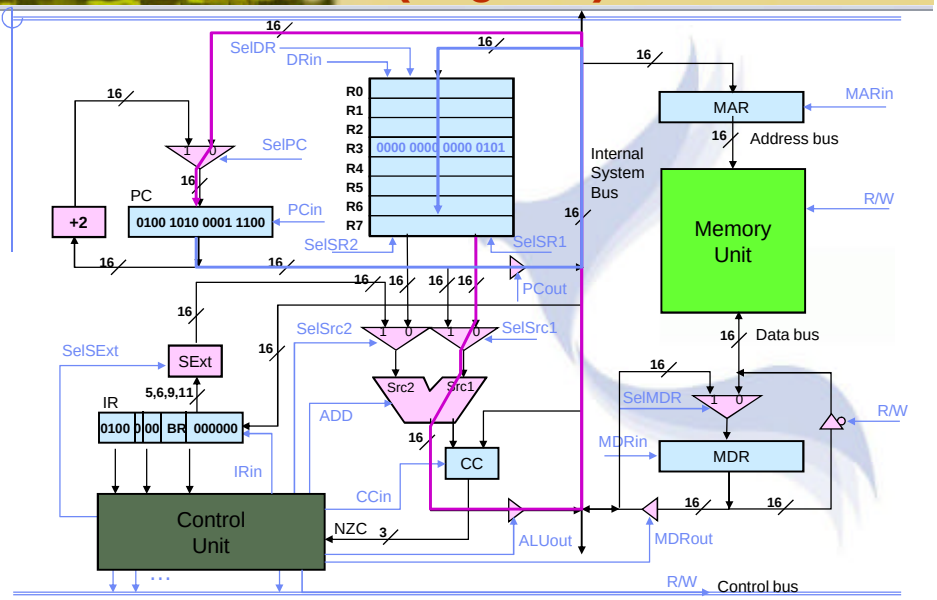




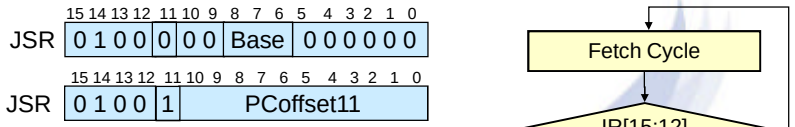
JMP (Register) Architecture



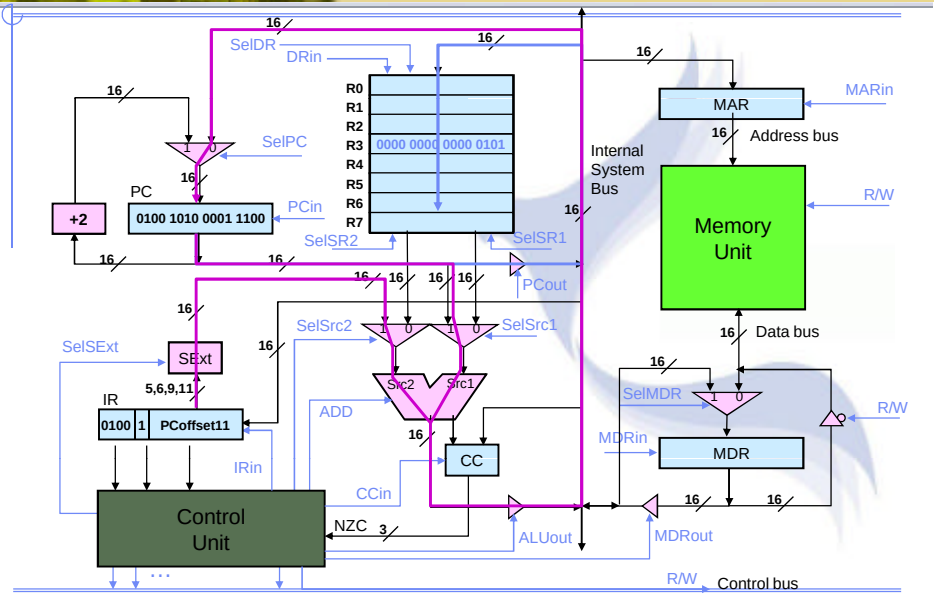
JSR (Register) Architecture



JSR (Register/ PC-Relative)



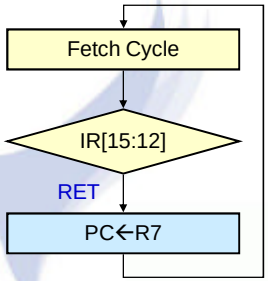
JSR (PC-Relative) Architecture



RET (Register)

RET

15 14 13 12	11 10 9	8 7 6	5 4 3 2 1 0
1 1 0 1	0 0 0	R7	0 0 0 0 0 0



TRAP

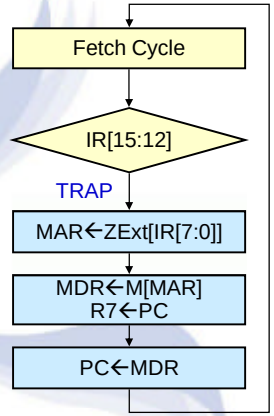
TRAP

15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
1 1 1 1	0 0 0 0	TrapVector8

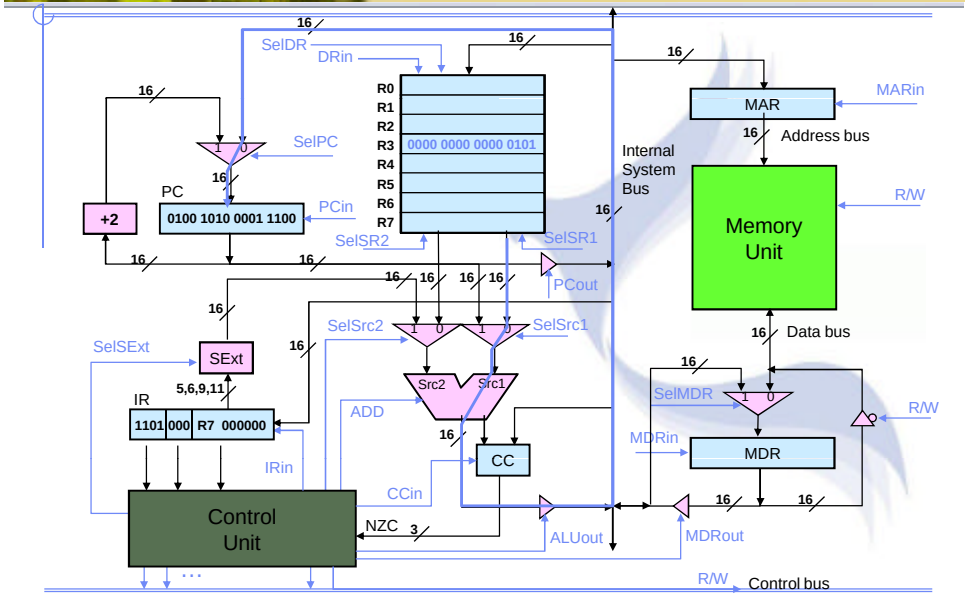
- Calls a service routine, identified by 8-bit "trap vector."

vector	routine
x23	input a character from the keyboard
x21	output a character to the monitor
x25	halt the program

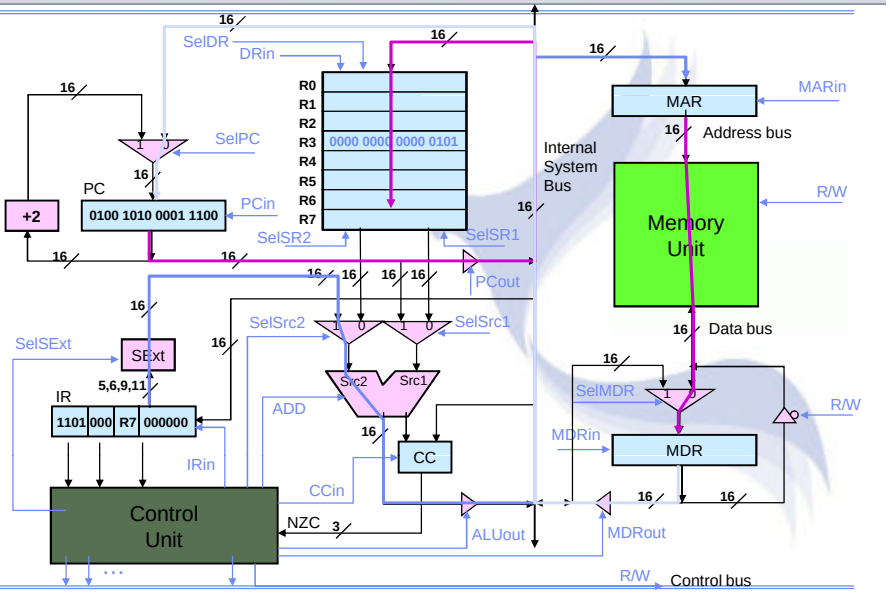
- When routine is done, PC is set to the instruction following TRAP.



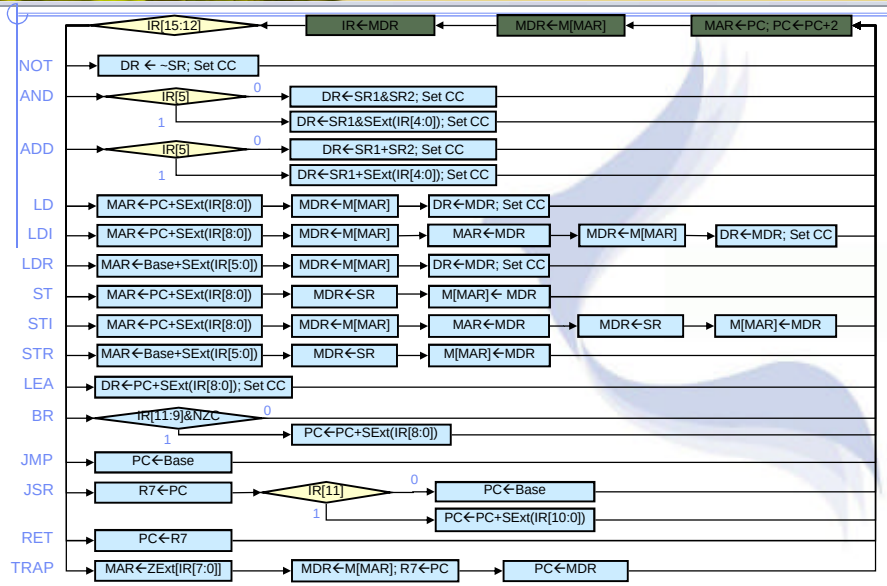
RET (Register) Architecture



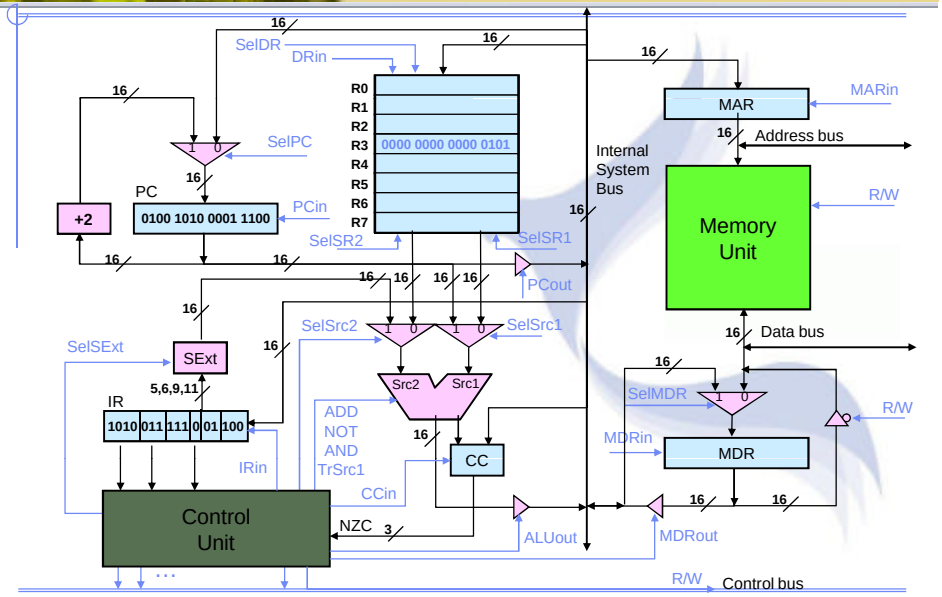
TRAP Architecture



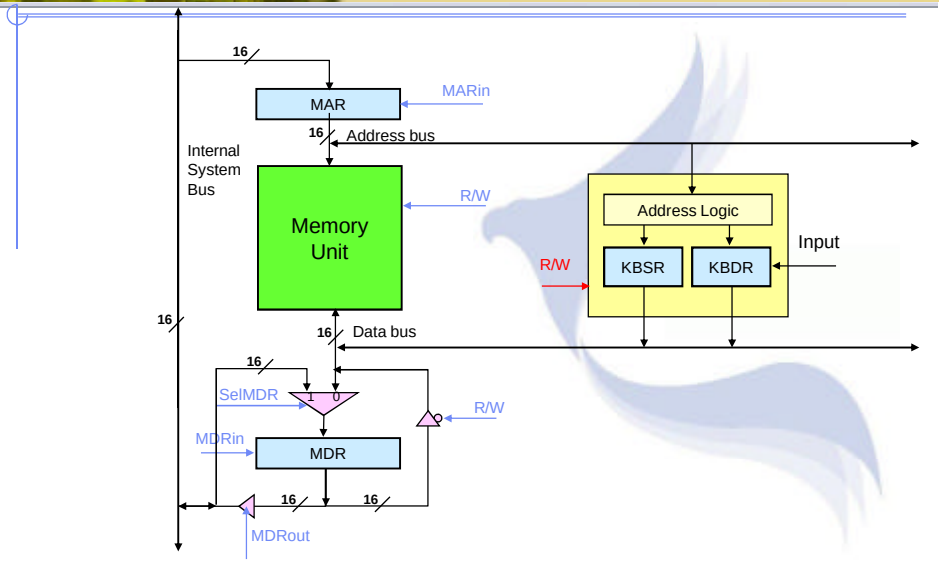
Complete RTL



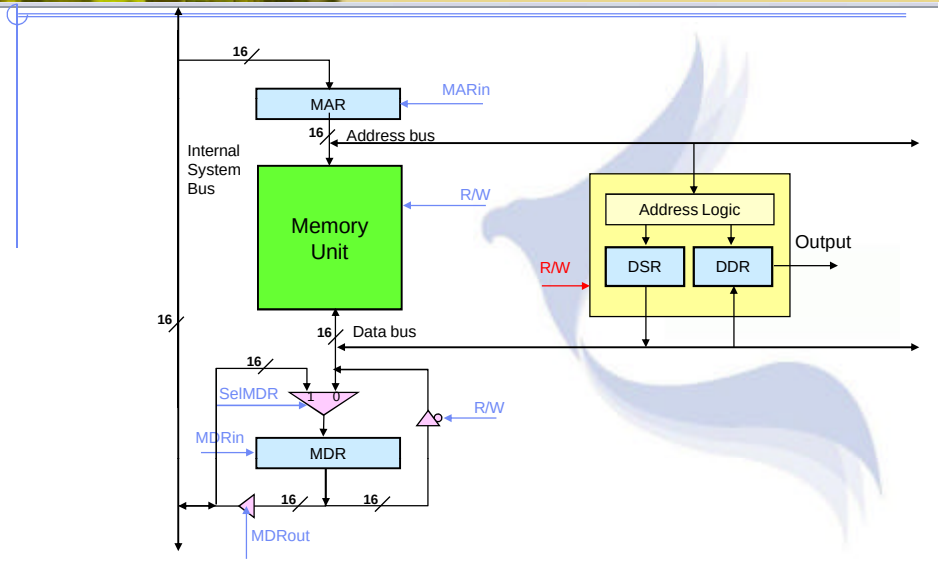
Complete Architecture



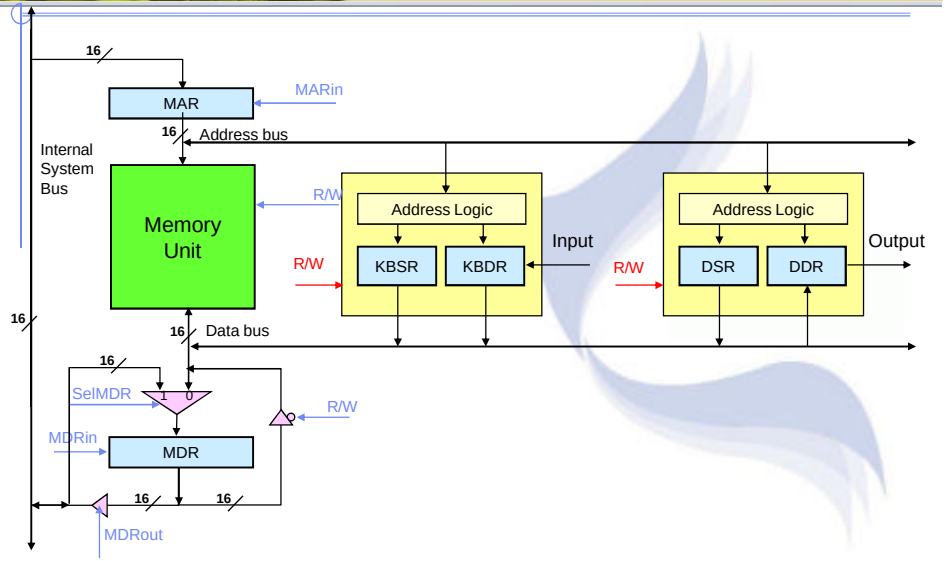
Memory-mapped Input



Memory-mapped Output



Memory-mapped I/O revisited



Simple Polling Routines

```
START   LDI     R1, A           ;Loop if Ready not set
        BRzp    START
        LDI     R0, B           ;If set, load char to R0
        BR      NEXT_TASK
A       .FILL   xFE00           ;Address of KBSR
B       .FILL   xFE02           ;Address of KBDR
```

Input a character from keyboard

```
START   LDI     R1, A           ;Loop if Ready not set
        BRzp    START
        STI     R0, B           ;If set, send char to DDR
        BR      NEXT_TASK
A       .FILL   xFE04           ;Address of DSR
B       .FILL   xFE06           ;Address of DDR
```

Output a character to the monitor

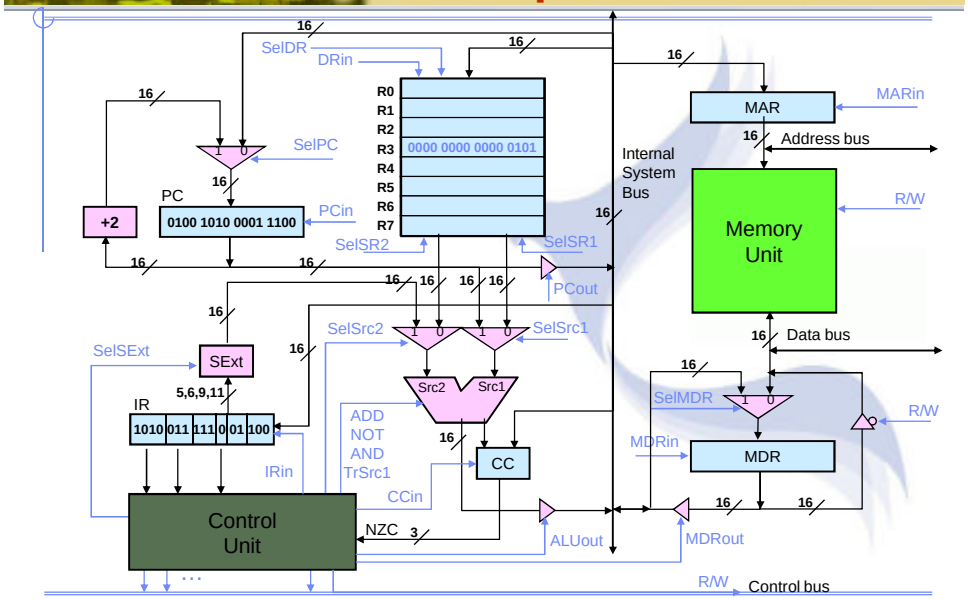
Keyboard Echo: combine the above

```
START   LDI     R1,KBSR         ;Loop if KB not ready
        BRzp    START
        LDI     R0,KBDR         ;Get character
ECHO    LDI     R1,CRTSR        ;Loop if monitor not ready
        BRzp    ECHO
        STI     R0,CRTDR        ;Send character
        BR      NEXT_TASK
KBSR    .FILL   xFE00           ;Address of KBSR
KBDR    .FILL   xFE02           ;Address of KBDR
DSR     .FILL   xFE04           ;Address of DSR
DDR     .FILL   xFE06           ;Address of DDR
```

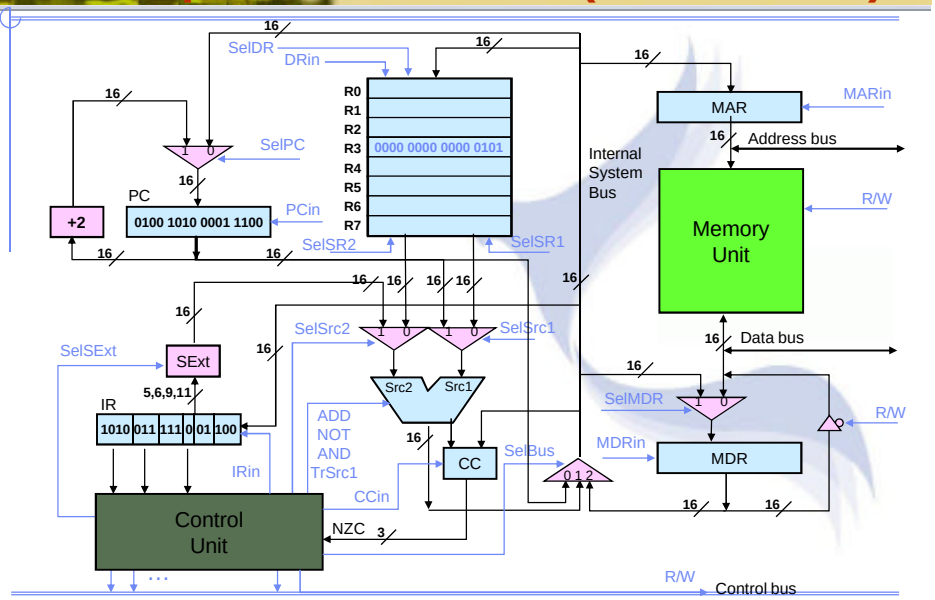

Example: Print a string

```
        LEA     R1, STR         ;Load address of string
LOOP    LDR     R0, R1, #0      ;get next char to R0
        BRZ     DONE            ;string ends with 0
LP2     LDI     R3, DSR         ;Loop until MON is ready
        BRzp    LP2
        STI     R0, DDR         ;Write next character
        ADD     R1, R1, #1  ;   Set address to next char
        BR      LOOP
STR     .STRINGZ   "Char String"
DONE    HALT
```

Complete Architecture



Complete Architecture (MUX version)



Finite State Machine