# Notes - Chapter 10

## Basic Computer Architectures

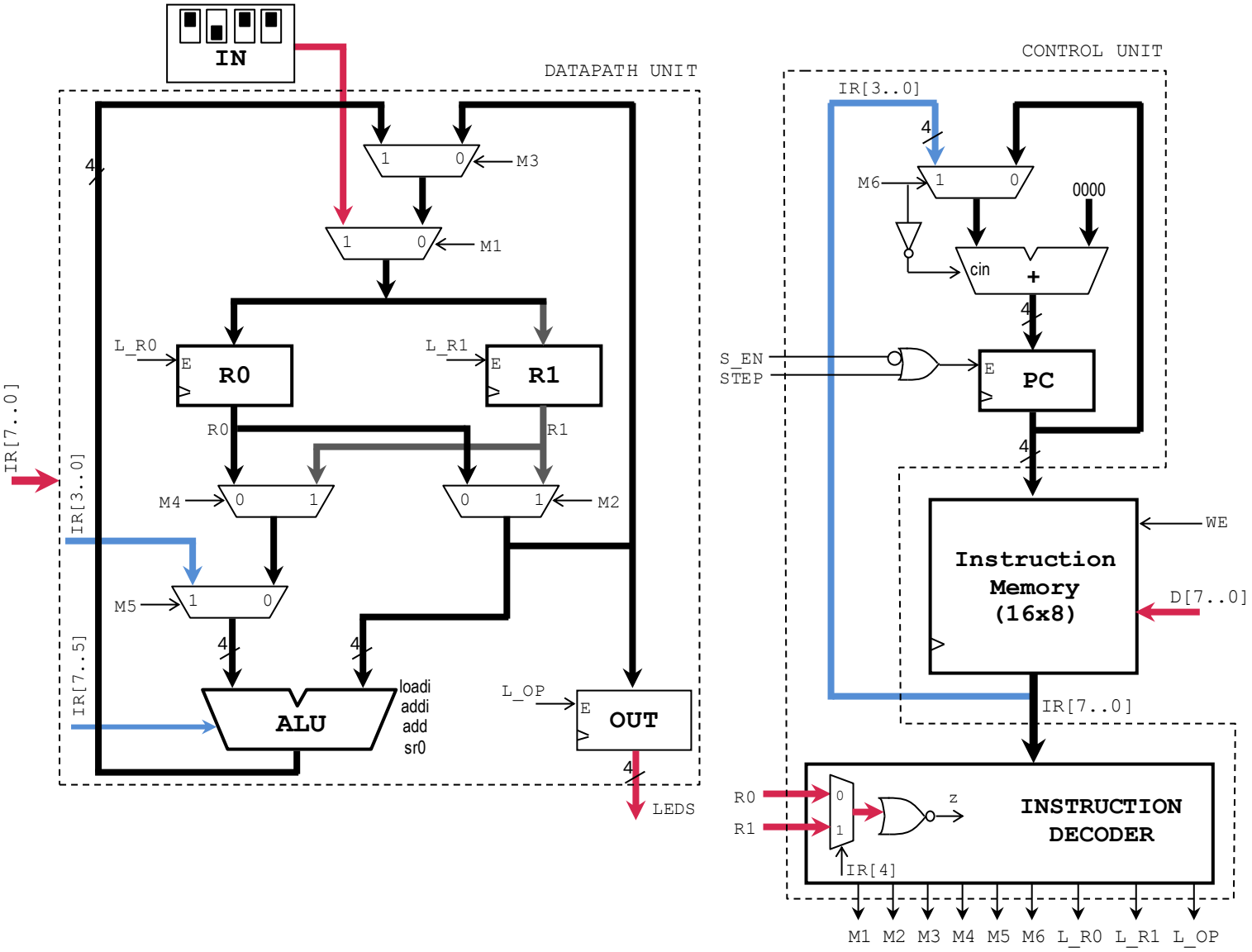
**Computer**: Processor + I/O + memory

**Harvard vs. Von Neumann**

*Harvard:*
- Instruction memory and data memory
- Operands usually placed in registers in the CPU: register-register architecture

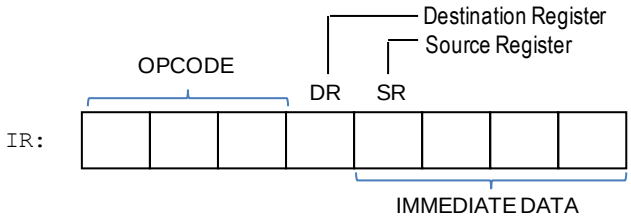
*Von Neumann:*
- One memory for both instruction and data
- Operands placed in a dedicated register called accumulator or in the instruction memory: register-memory architecture

**VCB1- computer: Harvard Computer:** Only one instruction memory. No data memory.



**Available Registers**:
`R0`(register 0), `R1`(register 1), `PC`(program counter), `IR`(instruction register), `OUT` (output register)

**Instruction Set**: Instructions are specified on the Instruction Register (`IR`):

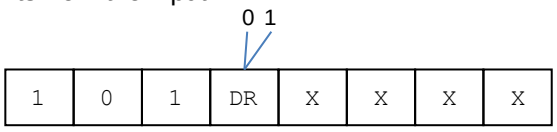



`DR=0` ⇒ `R0` is the destination register, `DR=1` ⇒ `R1` is the destination register.
`SR=0` ⇒ `R0` is the source register, `SR=1` ⇒ `R1` is the source register.
`OPCODE: IR[7..5]` `DATA: IR[3..0]`

| OPCODE (`IR[7..5]`) | Instruction | Operation |
|---|---|---|
| 000 | MOV DR, SR | DR ← SR |
| 001 | LOADI DR, DATA | DR ← DATA, DATA = IR[3..0] |
| 010 | ADD DR, SR | DR ← DR + SR |
| 011 | ADDI DR, DATA | DR ← DR + DATA, DATA = IR[3..0] |
| 100 | SR0 DR, SR | DR ← 0&SR[3..1] |
| 101 | IN DR | DR ← IN |
| 110 | OUT DR | OUT ← DR |
| 111 | JNZ DR, ADDRESS | PC ← PC + 1 if DR=0<br>PC ← IR[3..0] if DR≠0<br>* ADDRESS = IR[3..0] |

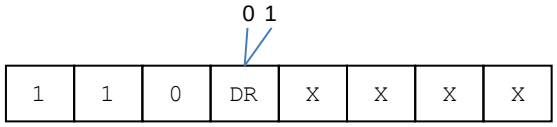
**Executing Instructions**:

✓ `IN DR:` `DR` grabs the contents from the input



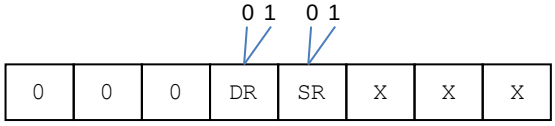

`IN R0: 1010XXXX ⇒ M1 ← 1, L_R0 ← 1, M6 ← 0`
`IN R1: 1011XXXX ⇒ M1 ← 1, L_R1 ← 1, M6 ← 0`

✓ `OUT DR:` Places the contents of `DR` on the output register




`OUT R0: 1100XXXX ⇒ M2 ← 0, L_OP ← 1, M6 ← 0`
`OUT R1: 1101XXXX ⇒ M2 ← 1, L_OP ← 1, M6 ← 0`

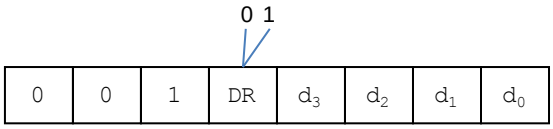
✓ `MOV DR, SR:` Copies the contents of `SR` onto `DR`




`MOV R0, R0: 00000XXX ⇒ M2 ← 0, M3 ← 0, M1 ← 0, L_R0 ← 1, M6 ← 0`
`MOV R1, R1: 00011XXX ⇒ M2 ← 1, M3 ← 0, M1 ← 0, L_R1 ← 1, M6 ← 0`
`MOV R0, R1: 00001XXX ⇒ M2 ← 1, M3 ← 0, M1 ← 0, L_R0 ← 1, M6 ← 0`
`MOV R1, R0: 00010XXX ⇒ M2 ← 0, M3 ← 0, M1 ← 0, L_R1 ← 1, M6 ← 0`

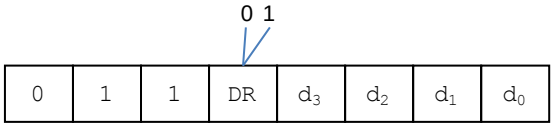
- "MOV R0,R0", "MOV R1,R1"(can be used as NOP instruction)

✓ `LOADI DR, DATA:` Copies immediate `DATA` onto `DR`



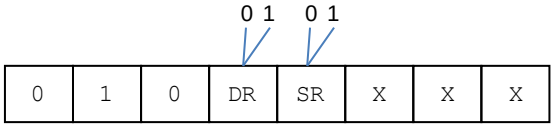

`LOADI R0, DATA:` $0010d_3d_2d_1d_0$ ⇒ `M5 ← 1, M3 ← 1, M1 ← 0, L_R0 ← 1, M6 ← 0`
`LOADI R1, DATA:` $0011d_3d_2d_1d_0$ ⇒ `M5 ← 1, M3 ← 1, M1 ← 0, L_R1 ← 1, M6 ← 0`

✓ `ADDI DR, DATA:` Adds immediate `DATA` and `DR`, and copies the result onto `DR`



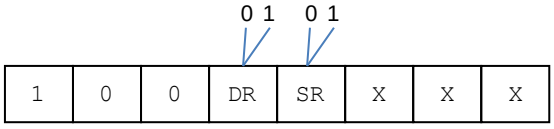

`ADDI R0, DATA:` $0110d_3d_2d_1d_0$ ⇒ `M2 ← 0, M5 ← 1, M3 ← 1, M1 ← 0, L_R0 ← 1, M6 ← 0`
`ADDI R1, DATA:` $0111d_3d_2d_1d_0$ ⇒ `M2 ← 1, M5 ← 1, M3 ← 1, M1 ← 0, L_R1 ← 1, M6 ← 0`

✓ `ADD DR, SR:` Adds `SR` and `DR`, and copies the result onto `DR`



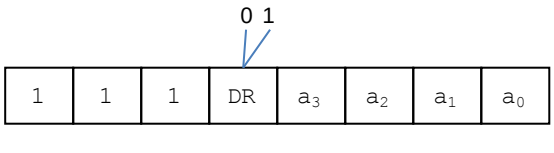

`ADD R0,R0: 01000XXX` ⇒ `M4←0, M5←0, M2←0, M3←1, M1←0, L_R0←1, M6←0`
`ADD R0,R1: 01001XXX` ⇒ `M4←0, M5←0, M2←1, M3←1, M1←0, L_R0←1, M6←0`
`ADD R1,R0: 01010XXX` ⇒ `M4←0, M5←0, M2←1, M3←1, M1←0, L_R1←1, M6←0`
`ADD R1,R1: 01011XXX` ⇒ `M4←1, M5←0, M2←1, M3←1, M1←0, L_R1←1, M6←0`

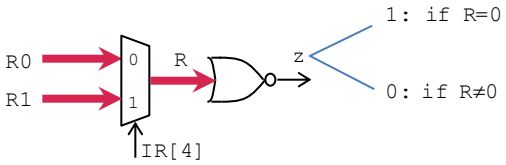
✓ `SR0 DR, SR:` Shifts (to the right) the contents of `SR` and places the result onto `DR`




`SR0 R0,R0: 10000XXX` ⇒ `M4←0, M5←0, M2←0, M3←1, M1←0, L_R0←1, M6←0`
`SR0 R0,R1: 10001XXX` ⇒ `M4←0, M5←0, M2←1, M3←1, M1←0, L_R0←1, M6←0`
`SR0 R1,R0: 10010XXX` ⇒ `M4←0, M5←0, M2←1, M3←1, M1←0, L_R1←1, M6←0`
`SR0 R1,R1: 10011XXX` ⇒ `M4←1, M5←0, M2←1, M3←1, M1←0, L_R1←1, M6←0`

✓ `JNZ DR, ADDRESS`




`JNZ R0, ADDRESS:` $1110a_3a_2a_1a_0$ ⇒ `M6 ← 0 if z = 1, M6 ← 1 if z = 0`
`JNZ R1, ADDRESS:` $1111a_3a_2a_1a_0$ ⇒ `M6 ← 0 if z = 1, M6 ← 1 if z = 0`

Keep in mind:

- The input `IR[7..5]` takes care automatically of the correct operation at the ALU.
- `M6 ← 0` means that `PC ← PC + 1`
- `M6 ← 1` means `PC ← IR[3..0]`