310-2202 โครงสร้างของระบบคอมพิวเตอร์ (Computer Organization)

Topic 5: The LC-3 Instructions: More Example

Damrongrit Setsirichok

Topic

- Addressing Mode Summary
- Another Example:

  Counting Occurrences of a Character
- Try Simulator

LC-3 ISA Group

Operate Instructions

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

Control Instructions

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
JSR	0	1	0	0	1	PCoffset11										
JSRR	0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP	1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
TRAP	1	1	1	1	0	0	0	0	TrapVector8							

Data Movement Instructions

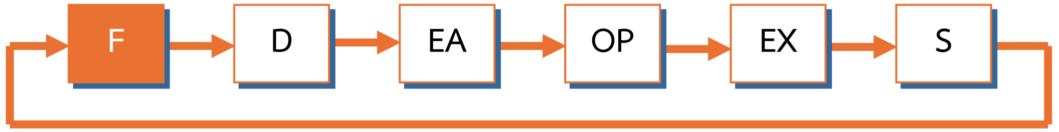
Load

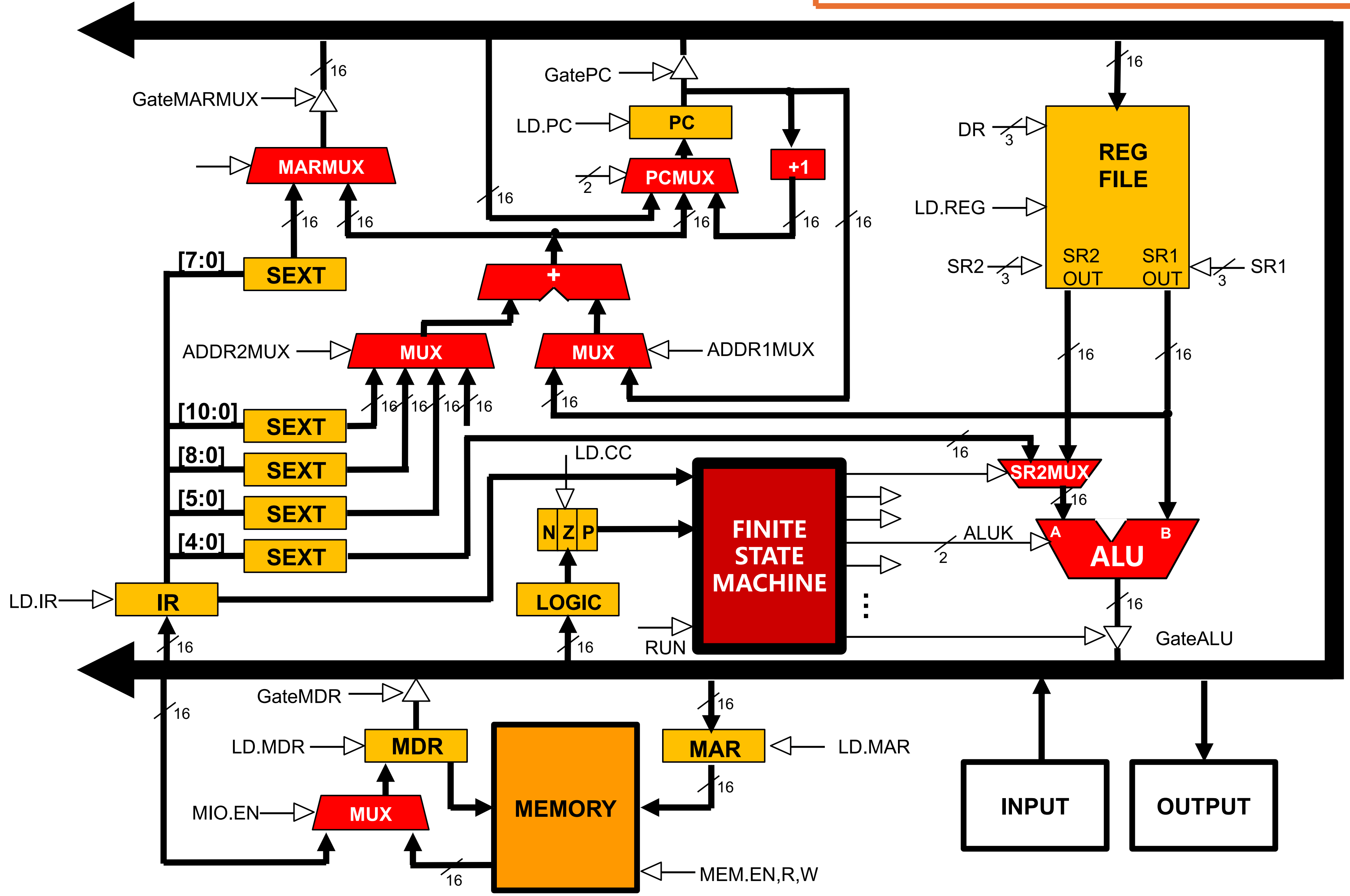
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

Store

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

Recall LC-3 Data Path





Addressing Mode Summary

Register:

- `R1 <- R1 + R2`
- `R1 <- NOT R2`

Immediate:

- `R1 <- R1 + (-2)`

Base + Offset:

- `R1       <- M[R2+4]`
- `M[R2+4] <- R1`

PC-Relative:

- `R1       <- M[PC+6]`
- `M[PC+6] <- R1`

Indirect:

- `R1          <- M[M[R2+4]]`
- `M[M[R2+4]] <- R1`

Condition Codes

- LC-3 has three condition code registers:

  N -- negative

  Z -- zero

  P -- positive (greater than zero)

- 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

Another Example:

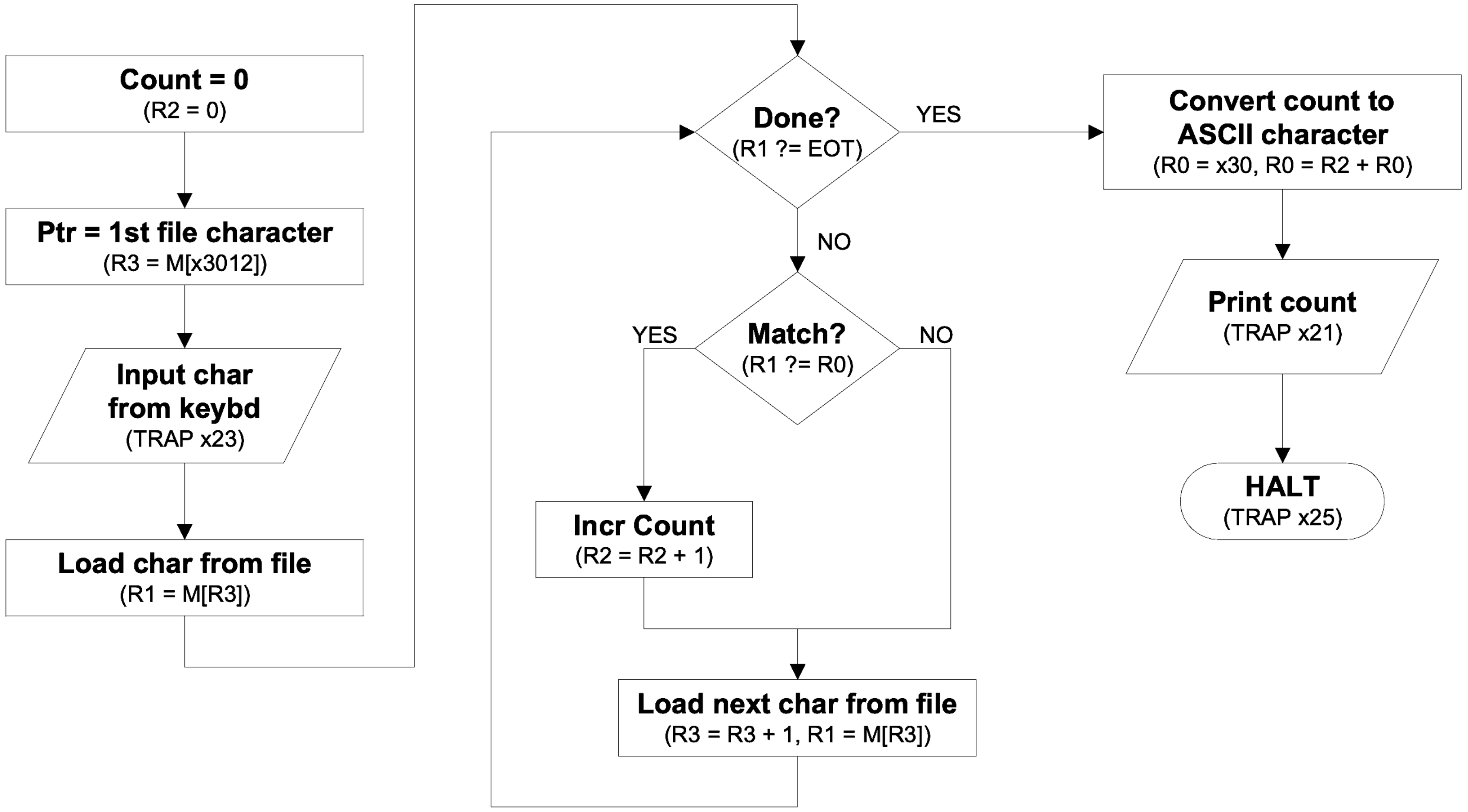
Counting Occurrences of a Character

Counting the occurrences of a character in a file

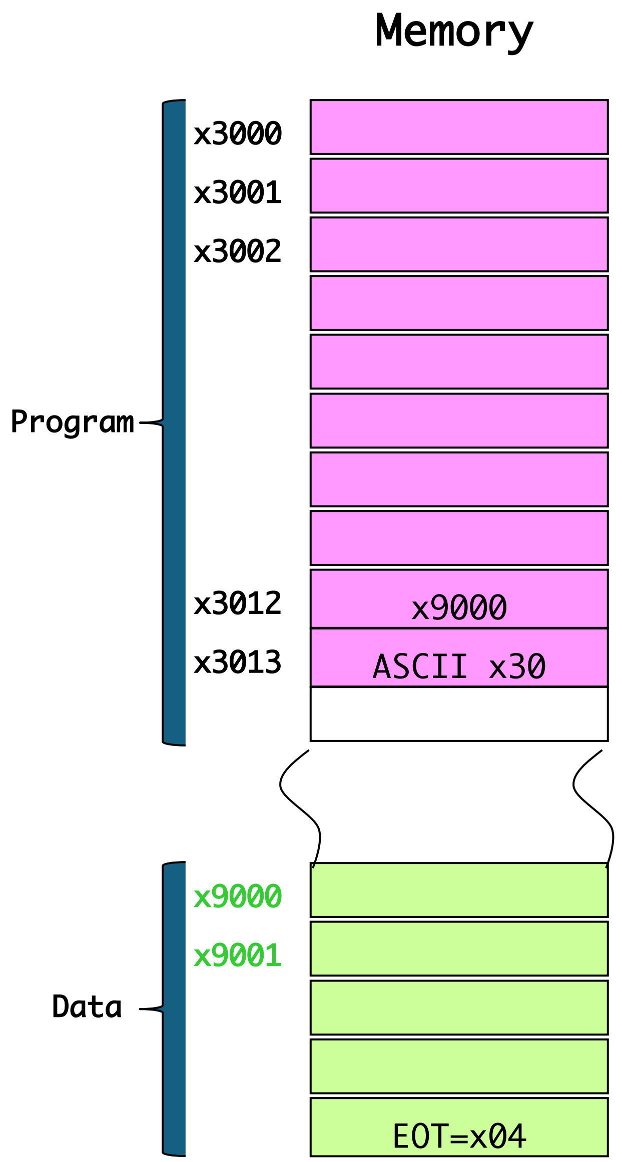
- Program begins at location x3000
- Read character from keyboard
- Load each character from a “file”
    - File is a sequence of memory locations
    - Starting address of file is stored in the memory location immediately after the program
- If file character equals input character, increment counter
- End of file is indicated by a special ASCII value: EOT (x04), called a sentinel
- At the end, print the number of characters and halt

(assume there will be less than 10 occurrences of the character)

Flow Chart



Count = 0
(R2 = 0)
Ptr = 1st file character
(R3 = M[x3012])
Input char from keybd
(TRAP x23)
Load char from file
(R1 = M[R3])
Done?
(R1 ?= EOT)
YES
NO
Match?
(R1 ?= R0)
YES
NO
Incr Count
(R2 = R2 + 1)
Load next char from file
(R3 = R3 + 1, R1 = M[R3])
Convert count to ASCII character
(R0 = x30, R0 = R2 + R0)
Print count
(TRAP x21)
HALT
(TRAP x25)



Memory
x3000
x3001
x3002
Program
x3012
x9000
x3013
ASCII x30
x9000
x9001
Data
EOT=x04

Register and Memory

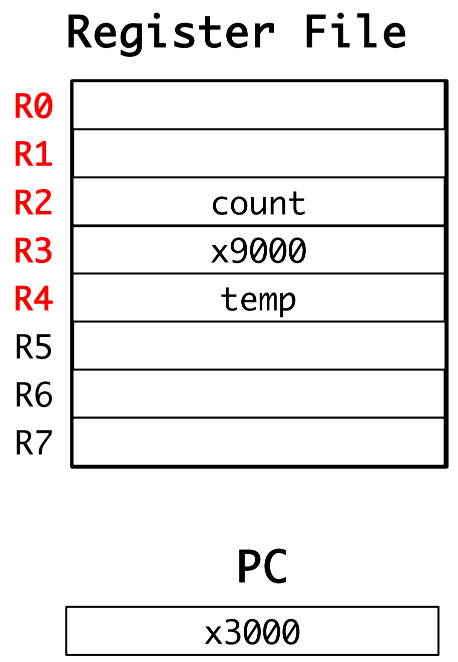
R0: hold the character that is being counted (typed from keyboard)

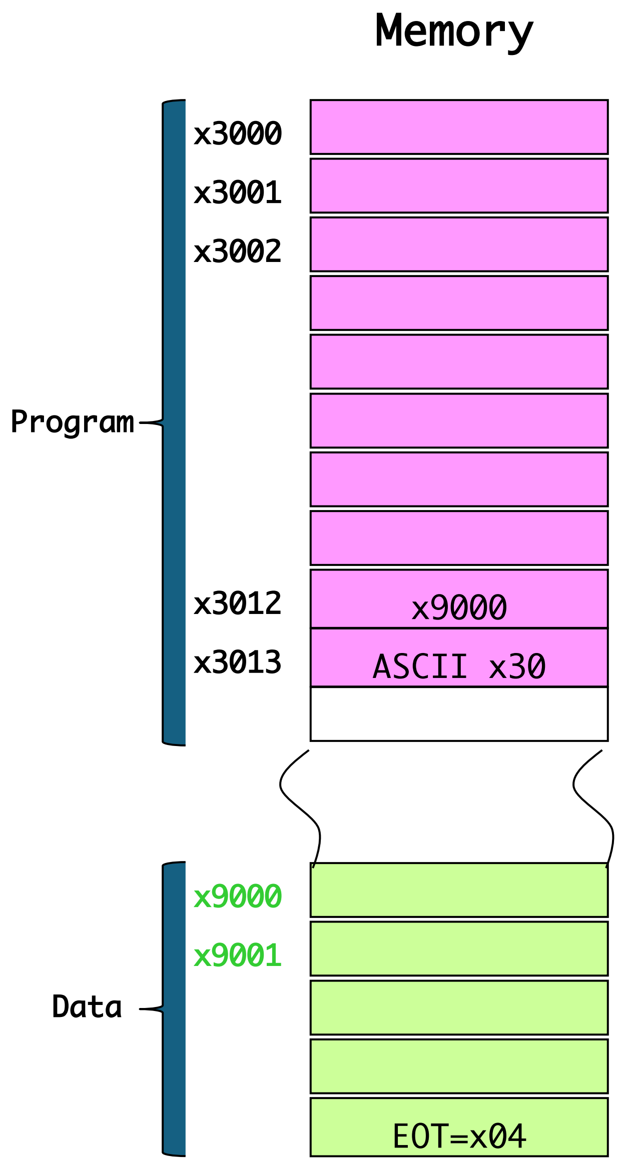
R1: hold, in turn, each character that we get from the file being examined

R2: keep track of the number of occurrences

R3: at first, M[x3012] =x9000

R4: temp, checking R4 = R1 - ASCII(EOT)





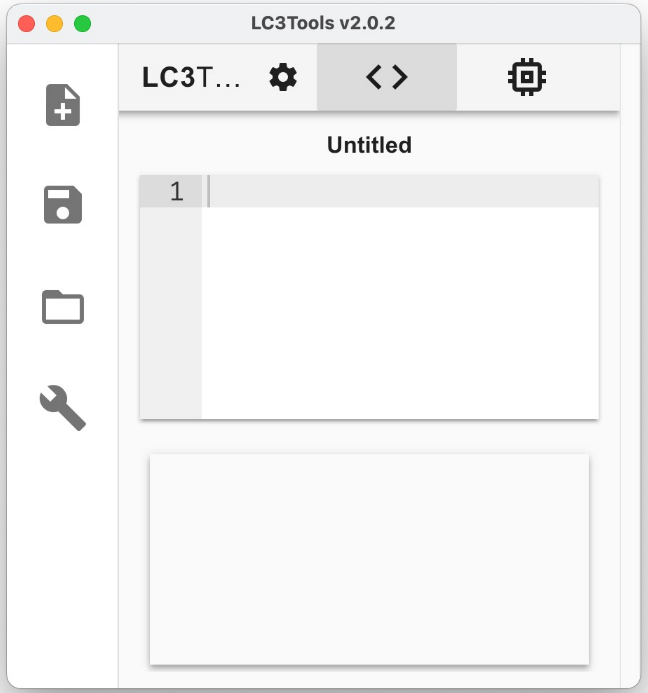
Machine Language Program

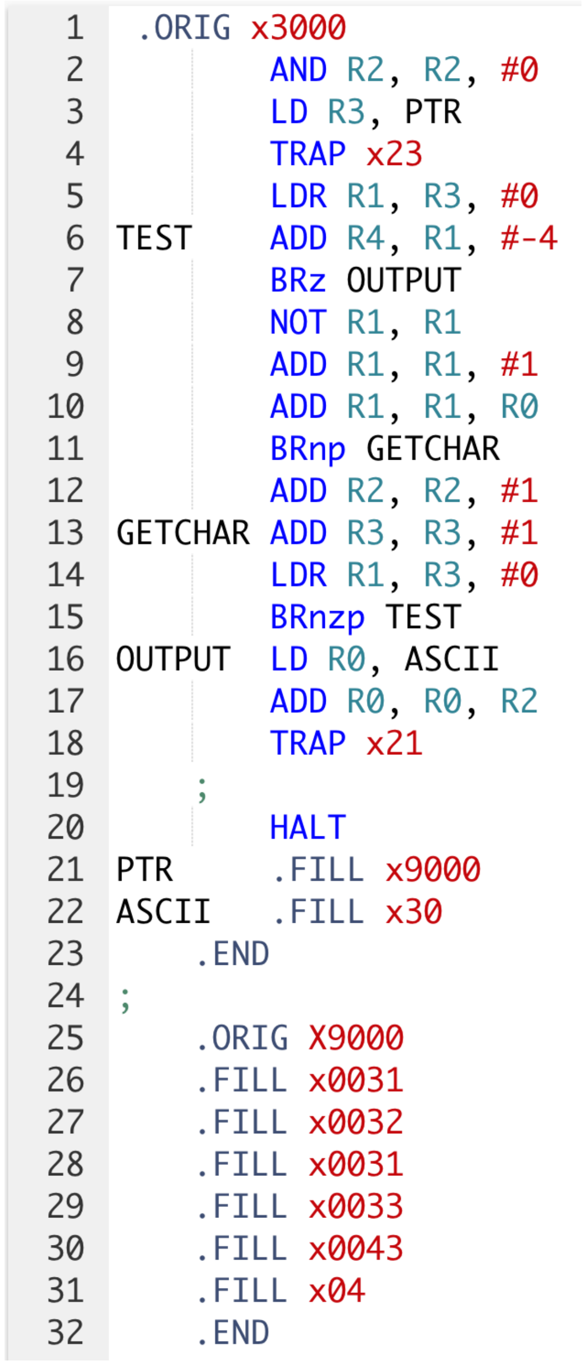
0000	BR
0001	ADD
0010	LD
0101	AND
1111	TRAP

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
x3000	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	`R2 <- 0`
x3001	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	`R3 <- M[x3012]`
x3002	1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	`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	1	1	1	1	1	0	0	`R4 <- R1-4`
x3005	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	`BRz 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	`BRnp x300B`
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	`BRnzp 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	`TRAP x21`
x3011	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	`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 TEMPLATE`

Assambly Program

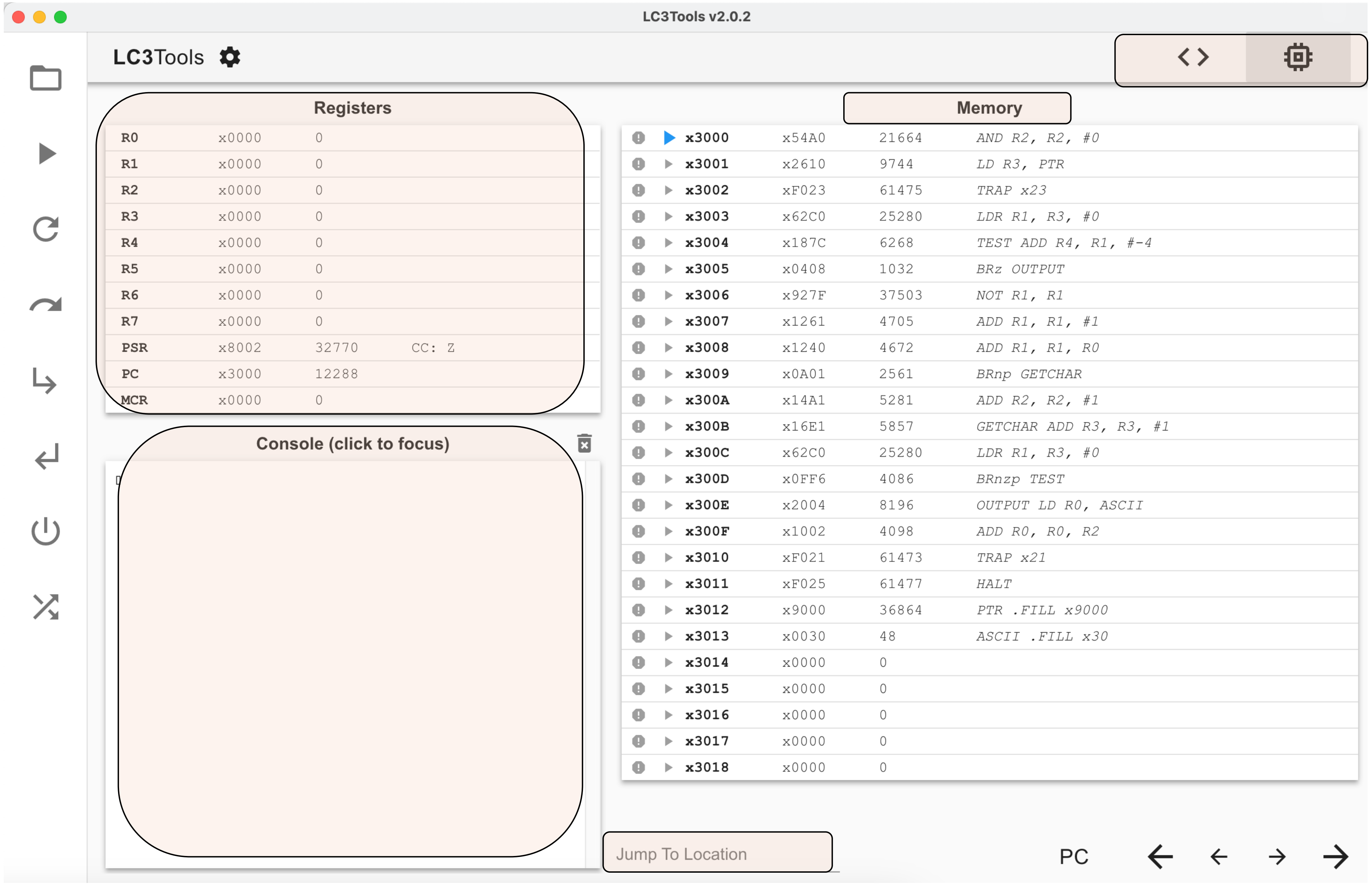
Try simulator





```
.ORIG x3000
        AND R2, R2, #0
        LD R3, PTR
        TRAP x23
        LDR R1, R3, #0
TEST    ADD R4, R1, #-4
        BRz OUTPUT
        NOT R1, R1
        ADD R1, R1, #1
        ADD R1, R1, R0
        BRnp GETCHAR
        ADD R2, R2, #1
GETCHAR ADD R3, R3, #1
        LDR R1, R3, #0
        BRnzp TEST
OUTPUT  LD R0, ASCII
        ADD R0, R0, R2
        TRAP x21
    ;
        HALT
PTR     .FILL x9000
ASCII   .FILL x30
    .END
;
    .ORIG X9000
    .FILL x0031
    .FILL x0032
    .FILL x0031
    .FILL x0033
    .FILL x0043
    .FILL x04
    .END
```

```
 .ORIG x3000
        AND R2, R2, #0
        LD R3, PTR
        TRAP x23
        LDR R1, R3, #0
TEST    ADD R4, R1, #-4
        BRz OUTPUT
        NOT R1, R1
        ADD R1, R1, #1
        ADD R1, R1, R0
        BRnp GETCHAR
        ADD R2, R2, #1
GETCHAR ADD R3, R3, #1
        LDR R1, R3, #0
        BRnzp TEST
OUTPUT  LD R0, ASCII
        ADD R0, R0, R2
        TRAP x21
    ;
        HALT
PTR     .FILL x9000
ASCII   .FILL x30
    .END
;
    .ORIG X9000
    .FILL x0031
    .FILL x0032
    .FILL x0031
    .FILL x0033
    .FILL x0043
    .FILL x04
    .END
```



```
R0     x0000   0
R1     x0000   0
R2     x0000   0
R3     x0000   0
R4     x0000   0
R5     x0000   0
R6     x0000   0
R7     x0000   0
PSR    x8002   32770   CC: Z
PC     x3000   12288
MCR    x0000   0
```

```
x3000   x54A0   21664   AND R2, R2, #0
x3001   x2610   9744    LD R3, PTR
x3002   xF023   61475   TRAP x23
x3003   x62C0   25280   LDR R1, R3, #0
x3004   x187C   6268    TEST ADD R4, R1, #-4
x3005   x0408   1032    BRz OUTPUT
x3006   x927F   37503   NOT R1, R1
x3007   x1261   4705    ADD R1, R1, #1
x3008   x1240   4672    ADD R1, R1, R0
x3009   x0A01   2561    BRnp GETCHAR
x300A   x14A1   5281    ADD R2, R2, #1
x300B   x16E1   5857    GETCHAR ADD R3, R3, #1
x300C   x62C0   25280   LDR R1, R3, #0
x300D   x0FF6   4086    BRnzp TEST
x300E   x2004   8196    OUTPUT LD R0, ASCII
x300F   x1002   4098    ADD R0, R0, R2
x3010   xF021   61473   TRAP x21
x3011   xF025   61477   HALT
x3012   x9000   36864   PTR .FILL x9000
x3013   x0030   48      ASCII .FILL x30
x3014   x0000   0
x3015   x0000   0
x3016   x0000   0
x3017   x0000   0
x3018   x0000   0
```

Program (1 of 2)

0000	BR
0001	ADD
0010	LD
0101	AND
1111	TRAP

Address	*Instruction*	*Comments*
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0 (counter)* *AND R2,R2, #0*
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	*R3 ← M[x3012] (ptr)* *LD R3, x3012 (LD R3, PTR)*
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	*Input to R0 (TRAP x23)* *TRAP x23 (GETC)*
TEST x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ← M[R3]* *LDR R1, R3, #0*
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	*R4 ← R1 – 4 (EOT)* *ADD R4,R1, #-4*
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	*If Z, goto x300E* *BR_z x300E (BR_z OUTPUT)*
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	*R1 ← NOT R1* *NOT R1,R1*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*R1 ← R1 + 1* *ADD R1,R1,#1*
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	*R1 ← R1 + R0* *ADD R1,R1,R0*
x3009	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1	*If N or P, goto x300B* *BR_{np} x300B (BR_{np} GETCHAR)*

Program (2 of 2)

0000	BR
0001	ADD
0010	LD
0101	AND
1111	TRAP

	Address	*Instruction*	*Comments*
	x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	*R2 ← R2 + 1* *ADD R2,R2,#1*
GETCHAR	x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	*R3 ← R3 + 1* *ADD R3,R3,#1*
	x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ← M[R3]* *LDR R1,R3,#0*
	x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	*Goto x3004* *BR$_{nzp}$ x3004 (BR$_{nzp}$ TEST)*
OUTPUT	x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0	*R0 ← M[x3013]* *LD R0,x3013 (LD R0, ASCII)*
	x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	*R0 ← R0 + R2* *ADD R0,R0,R2*
	x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	*Print R0* *TRAP x21 (OUT)*
	x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	*HALT* *TRAP x25 (HALT)*
	X3012	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	Starting Address of File (X9000)
	x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	*ASCII x30 (‘0’)*

Instruction Set Architecture (ISA) Evolution

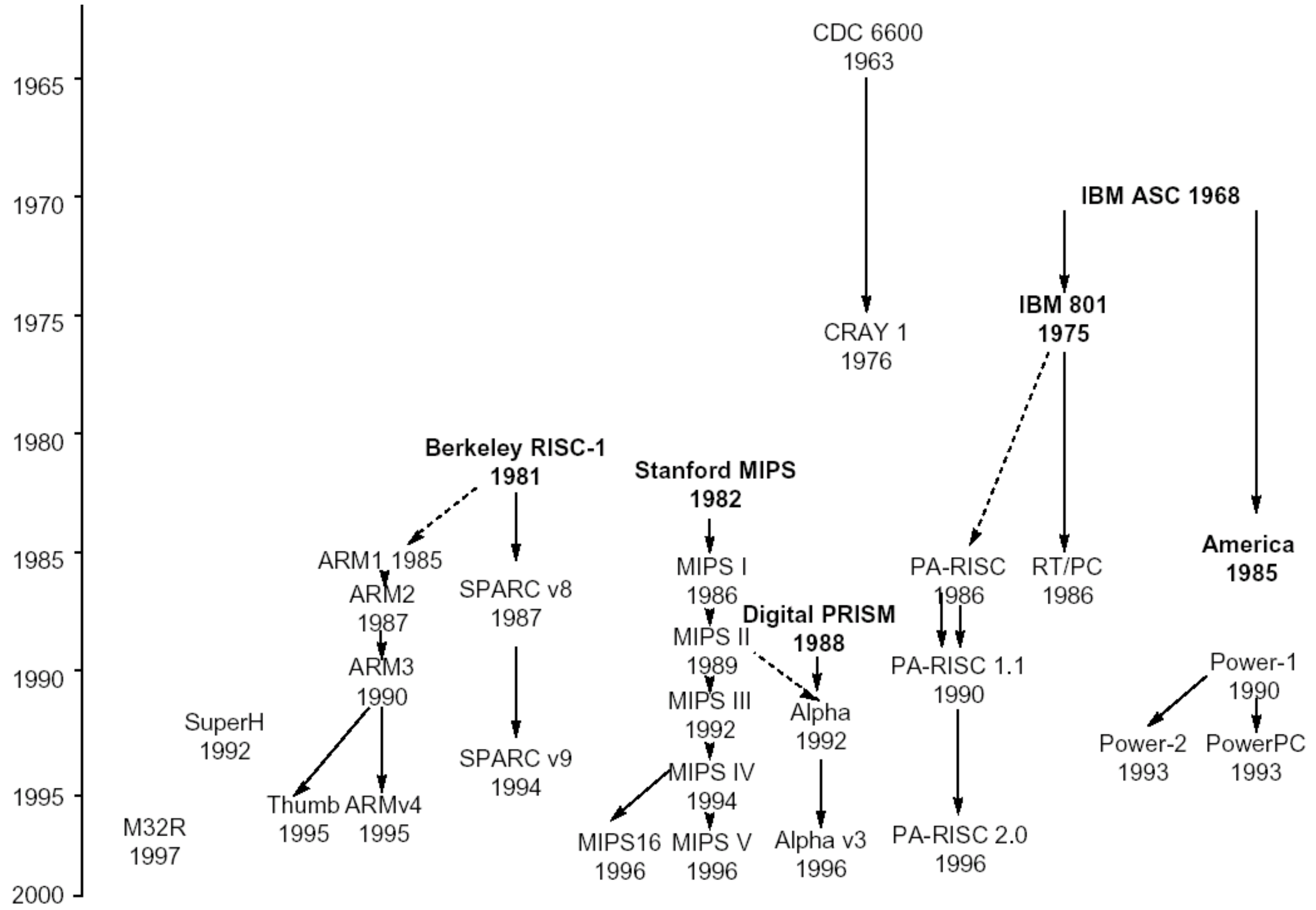
Instruction Set Architecture (ISA)

- Computer's native operations called **instructions.**
- Job of a CPU (Central Processing Unit, a.k.a. Core): execute instructions
    - Instructions: CPU's primitives operations
    - Instructions performed **one after another** in sequence
    - Each instruction does a small amount of work (a tiny part of a larger program)
    - Each instruction has an operation applied to operands, and might be used to change the sequence of instruction
- **Instruction set architecture (ISA)** specifies the **set of commands (instructions)** a computer can execute
- Hardware registers provide a few very fast variables for instructions to operate on

Instruction Set Architecture (ISA)

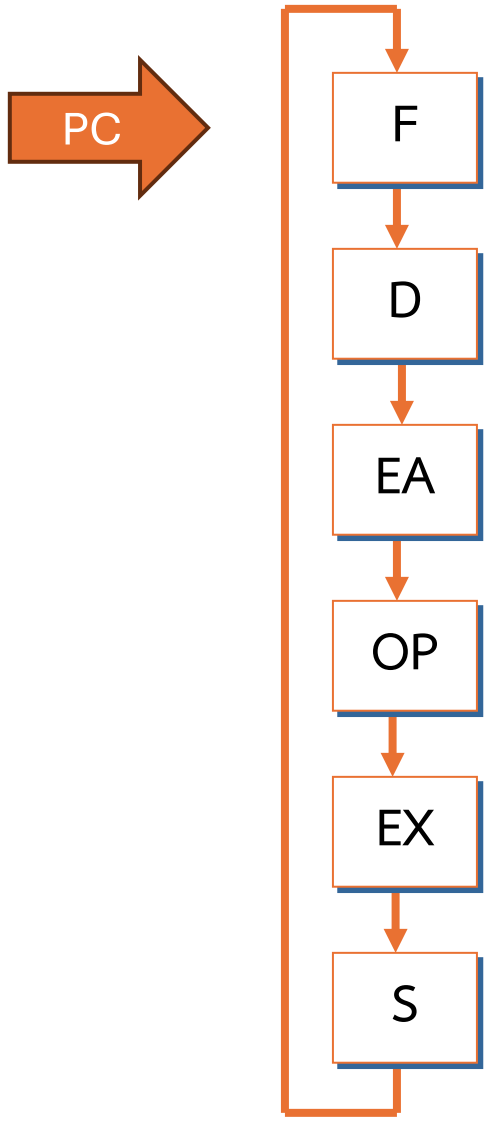
- The instruction set defines all the valid instructions.
- CPUs belong to **“families”** each implementing its own set of instructions
- CPU’s particular set of instructions implements an Instruction Set Architecture (ISA)
- Examples:
    - ARM,
    - Intel x86
    - MIPS
    - RISC-V
    - IBM/Motorola PowerPC (old Mac)
    - Intel IA64,
    - …

Instruction set architecture evolution



Summary

Instruction Processing: Finite State Automata



702 **appendix C** The Microarchitecture of the LC-3

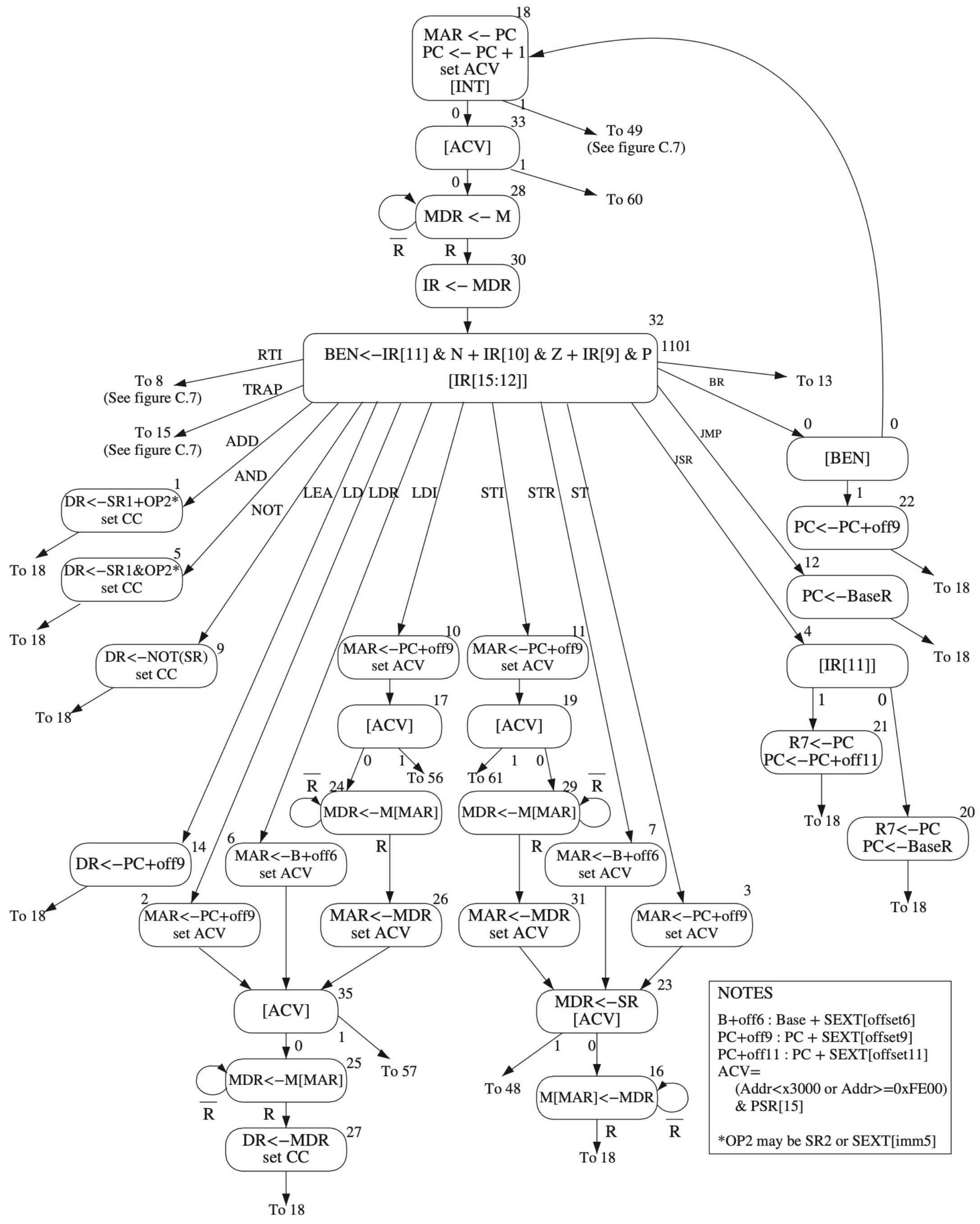
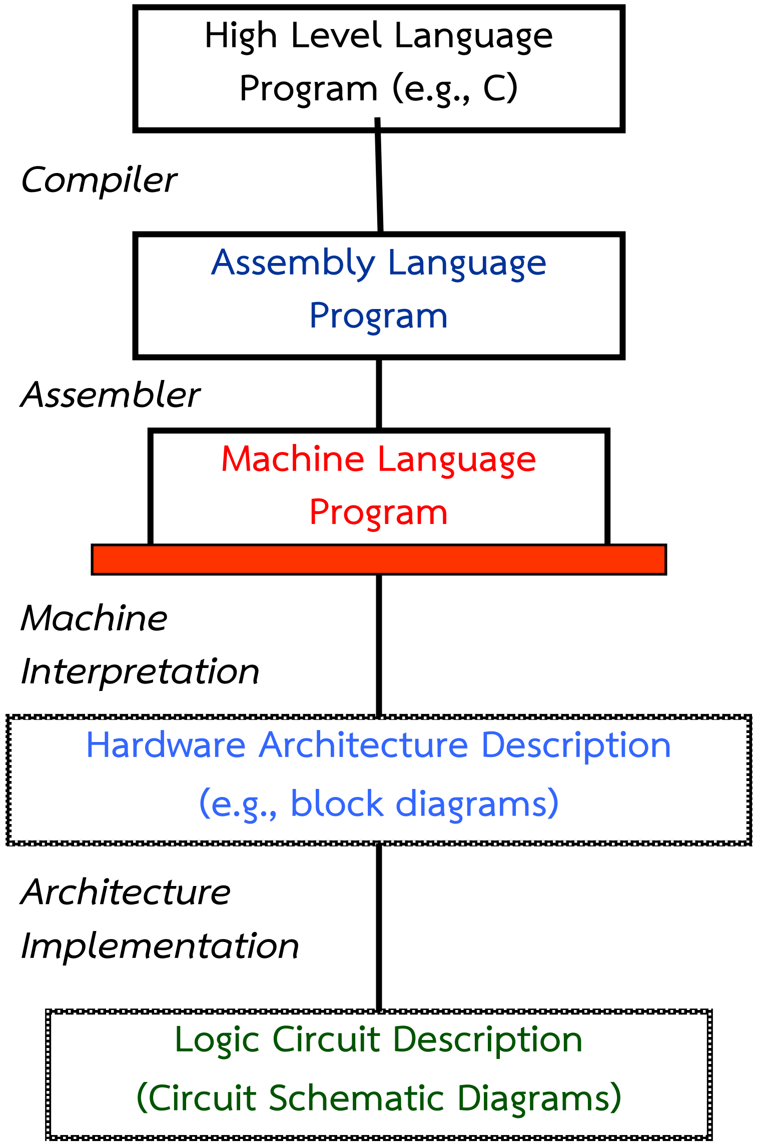


Figure C.2 **A state machine for the LC-3.**

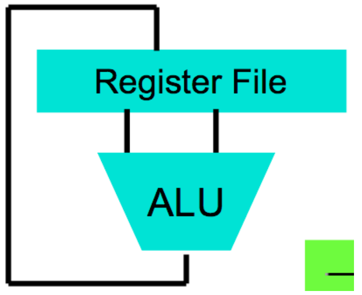
How do we get the electrons to do the work?

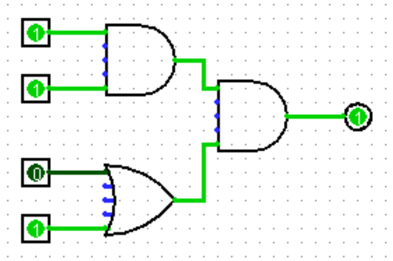

High Level Language Program (e.g., C)
Compiler
Assembly Language Program
Assembler
Machine Language Program
Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)
Architecture Implementation
Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $t0, 0(a0)
lw  $t1, 4(a0)
sw  $t1, 0(a0)
sw  $t0, 4(a0)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```





LC-3 ISA Group

Operate Instructions

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	DR			SR1			0	0	0	SR2		
ADD	0	0	0	1	DR			SR1			1	Imm5				
AND	0	1	0	1	DR			SR1			0	0	0	SR2		
AND	0	1	0	1	DR			SR1			1	Imm5				
NOT	1	0	0	1	DR			SR1			1	1	1	1	1	1
Reserved	1	1	0	1												

Control Instructions

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR	0	0	0	0	n	z	p	PCoffset9								
JSR	0	1	0	0	1	PCoffset11										
JSRR	0	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RTI	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
JMP	1	1	0	0	0	0	0	BaseR			0	0	0	0	0	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
TRAP	1	1	1	1	0	0	0	0	TrapVector8							

Data Movement Instructions

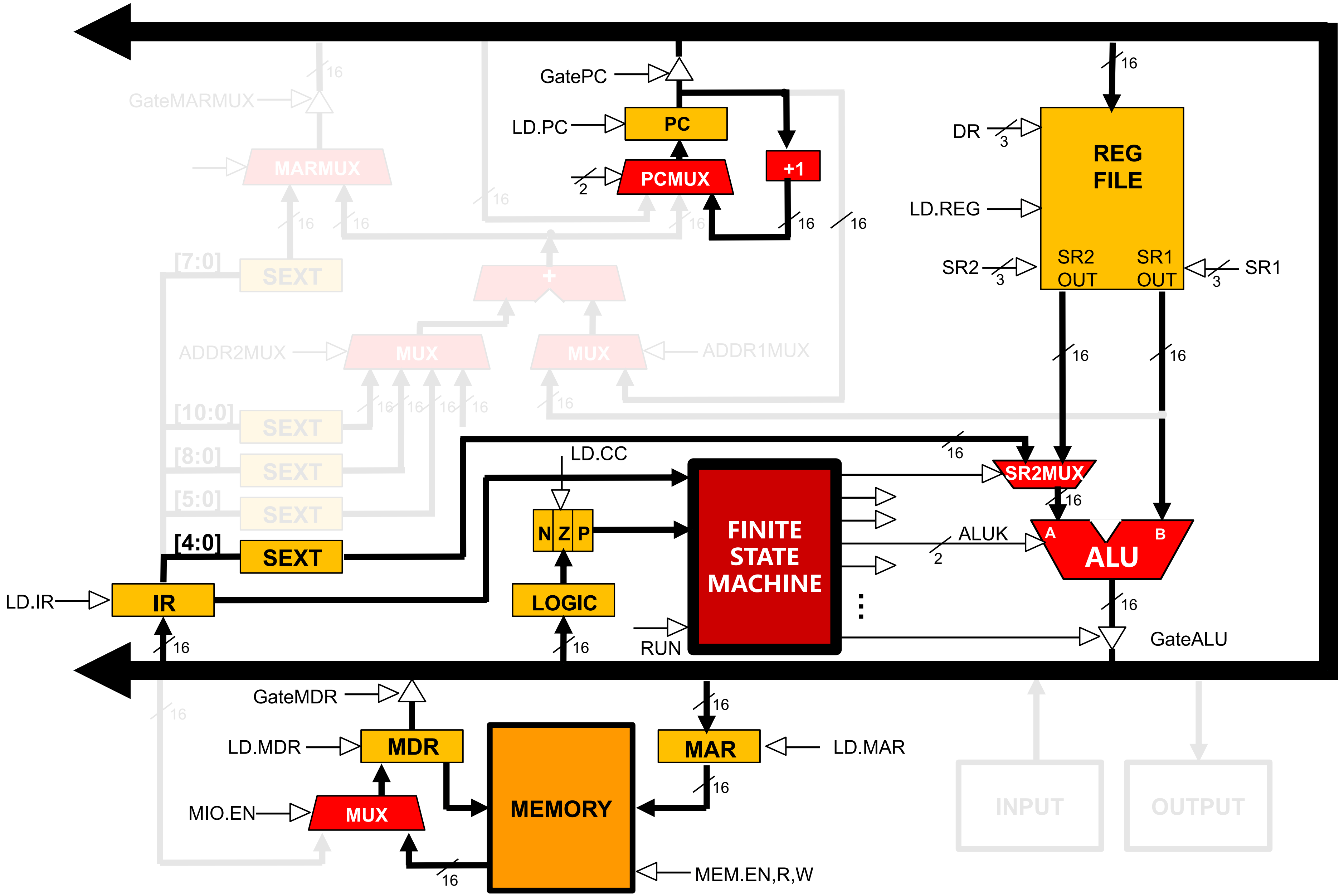
Load

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LD	0	0	1	0	DR			PCoffset9								
LDR	0	1	1	0	DR			BaseR			PCoffset6					
LDI	1	0	1	0	DR			PCoffset9								
LEA	1	1	1	0	DR			PCoffset9								

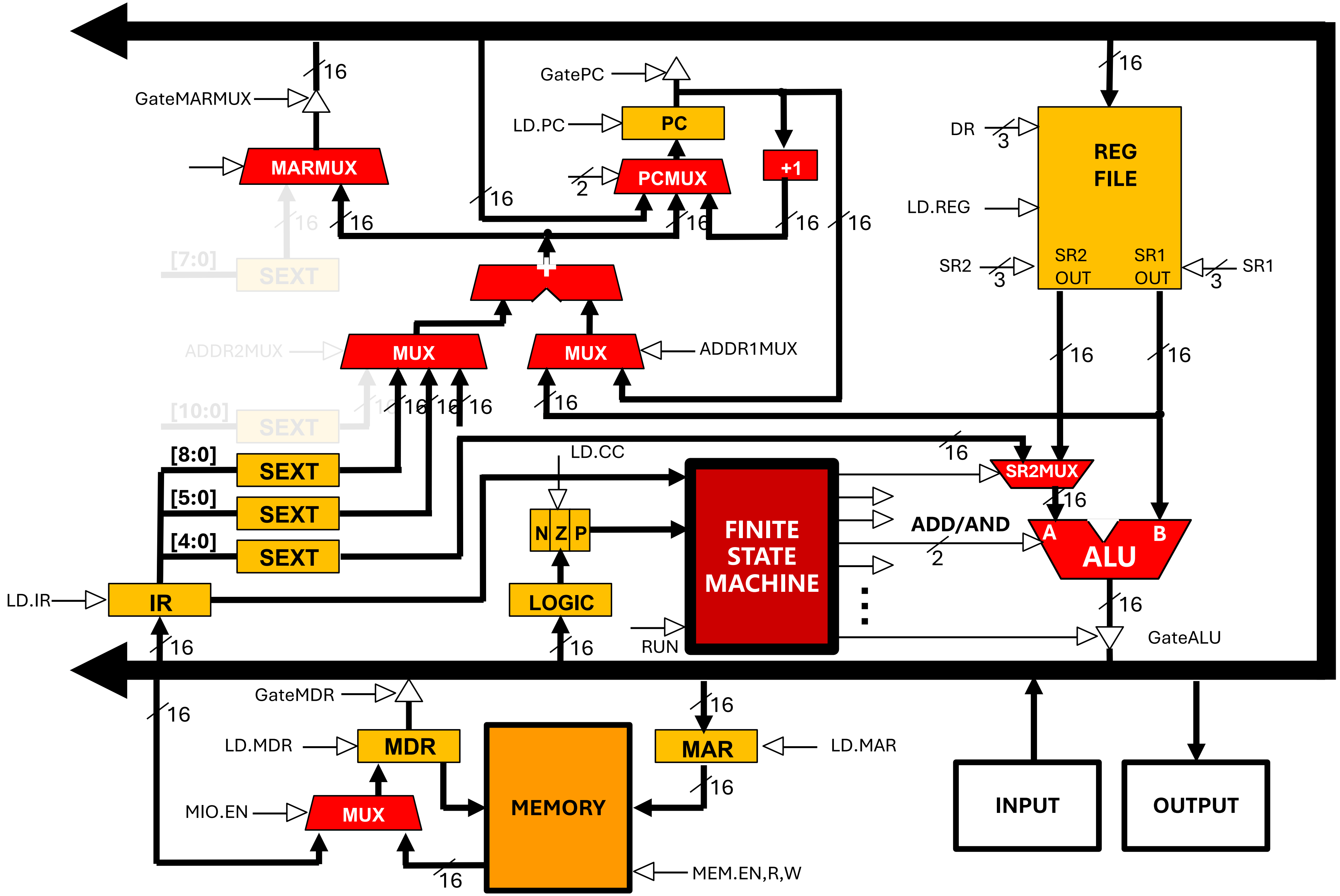
Store

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST	0	0	1	1	SR			PCoffset9								
STR	0	1	1	1	SR			BaseR			PCoffset6					
STI	1	0	1	1	SR			PCoffset9								

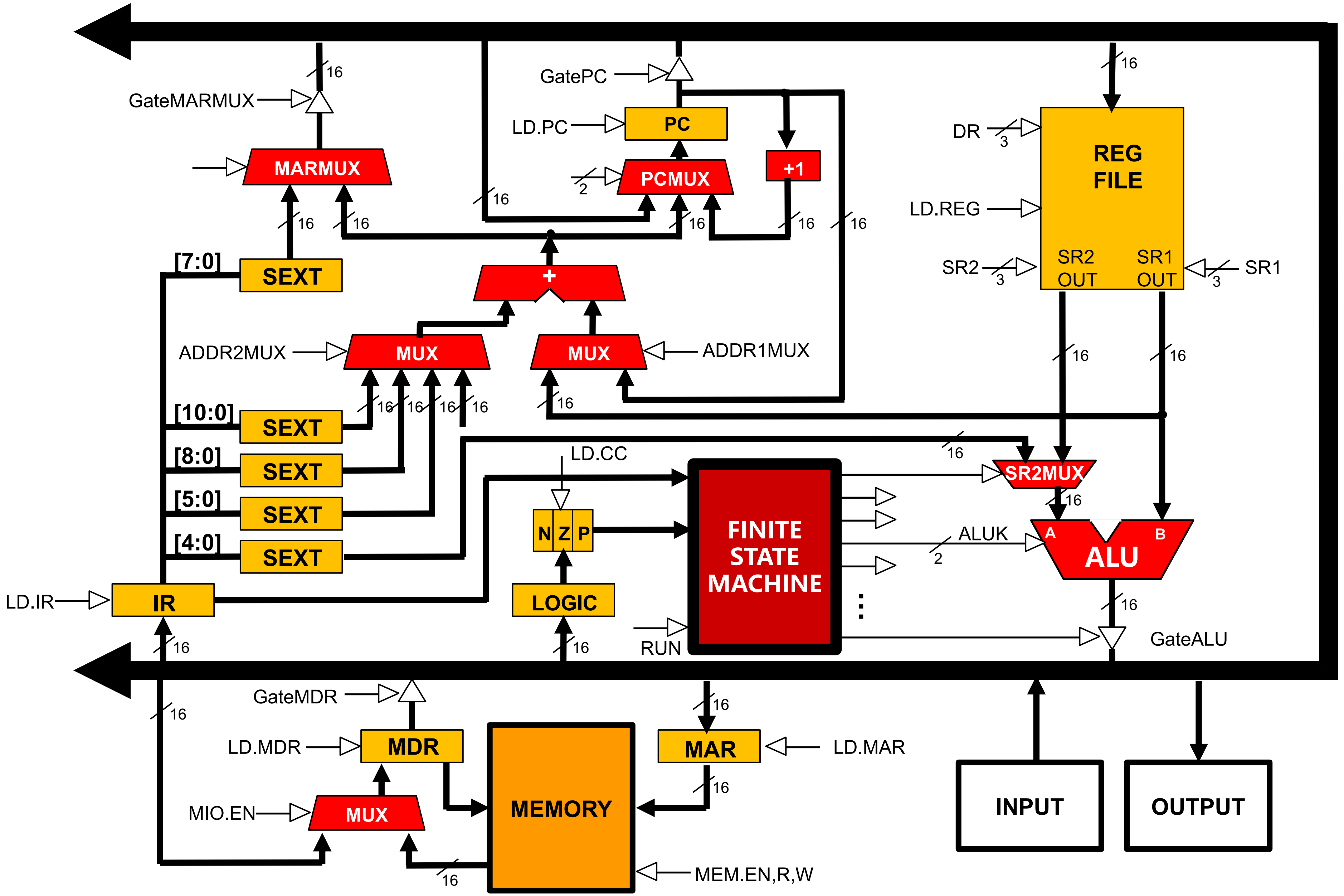
LC-3 Data Path After Operate Instruction



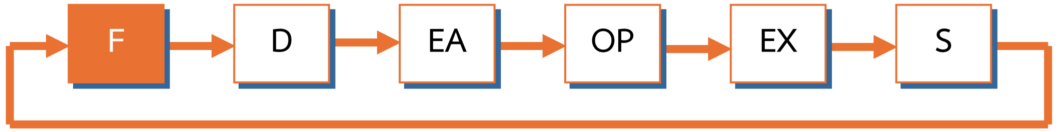
LC-3 Data Path After Load/Store Instruction

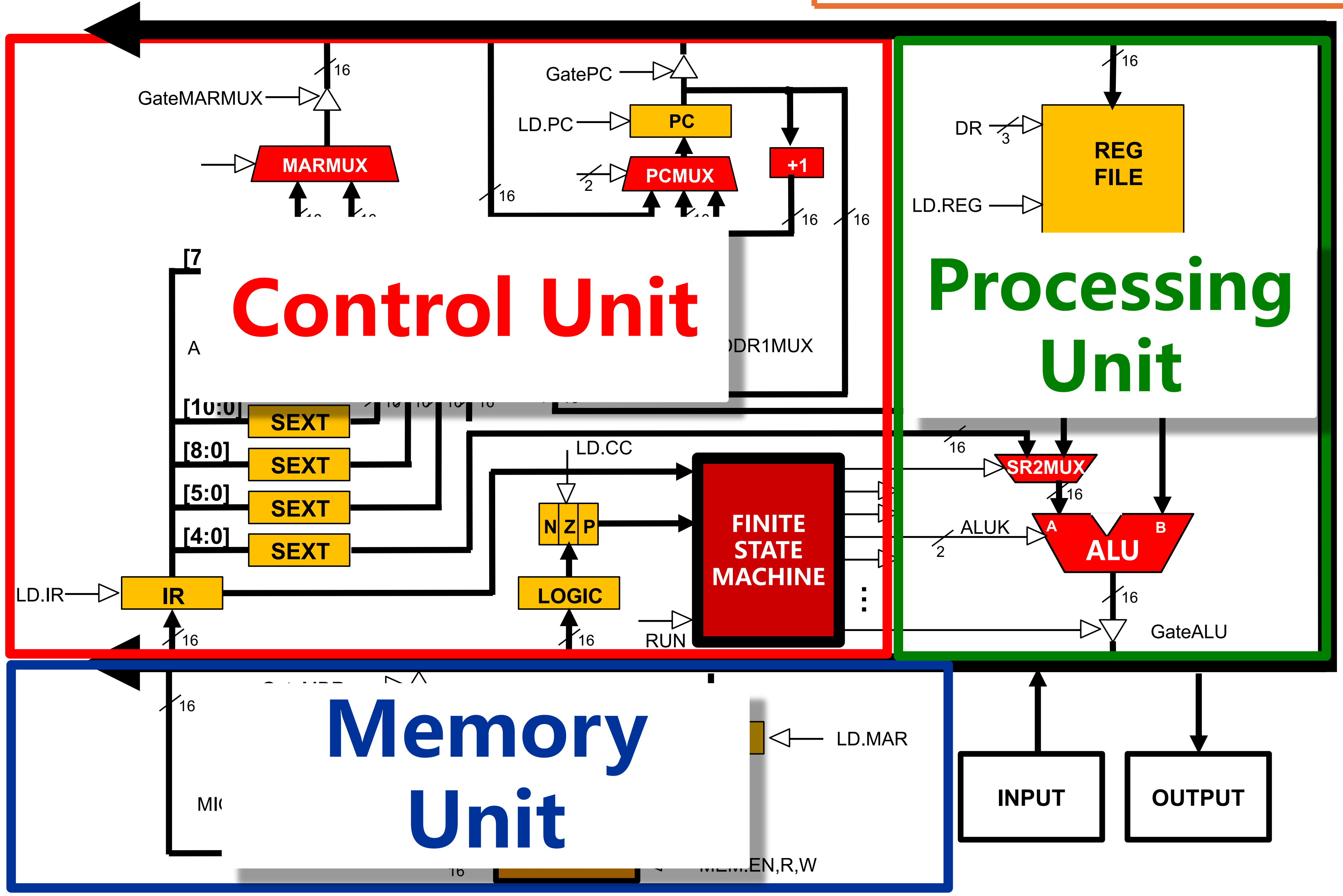


LC-3 Data Path After Control Instruction

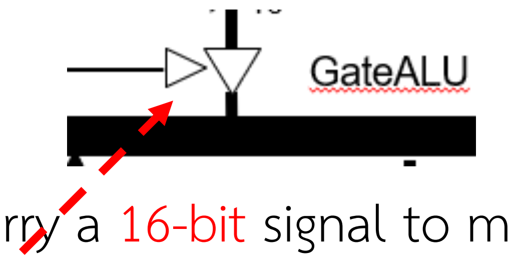


Recall LC-3 Data Path





Data Path Components



Global bus

- special set of wires that carry a 16-bit signal to many components
- inputs to the bus are "tri-state devices" that only place a signal on the bus when they are enabled
- only one (16-bit) signal should be enabled at any time
  - control unit decides which signal "drives" the bus
- any number of components can read the bus
  - register only captures bus data if it is write-enabled by the control unit

Memory

- Control and data registers for memory and I/O devices
- memory: MAR, MDR (also control signal for read/write)

Data Path Components

ALU

- Accepts inputs from register file
  and from sign-extended bits from IR (immediate field).
- Output goes to bus.
  - used by condition code logic, register file, memory

Register File

- Two read addresses (SR1, SR2), one write address (DR)
- Input from bus
  - result of ALU operation or memory read
- Two 16-bit outputs
  - used by ALU, PC, memory address
  - data for store instructions passes through ALU

Data Path Components

PC and PCMUX

- Three inputs to PC, controlled by PCMUX
  1. PC+1 – FETCH stage
  2. Address adder – BR, JMP
  3. bus – TRAP

MAR and MARMUX

- Two inputs to MAR, controlled by MARMUX
  1. Address adder – LD/ST, LDR/STR
  2. Zero-extended IR[7:0] -- TRAP

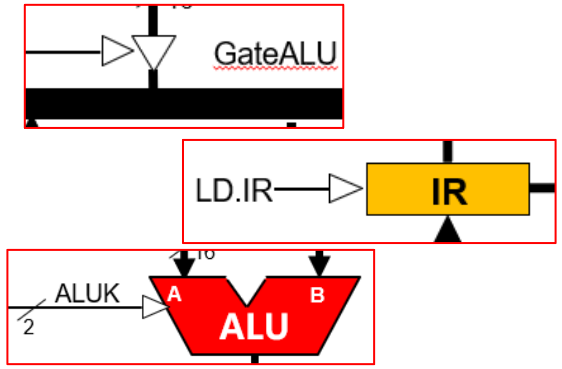
Data Path Components

Condition Code Logic

- Looks at value on bus and generates N, Z, P signals
- Registers set only when control unit enables them (LD.CC)
    - only certain instructions set the codes
      (ADD, AND, NOT, LD, LDI, LDR, LEA)

Control Unit – Finite State Machine (FSM)

- On each machine cycle, changes control signals for next phase of instruction processing
    - who drives the bus? (GatePC, GateALU, •••)
    - which registers are write enabled? (LD.IR, LD.REG, •••)
    - which operation should ALU perform? (ALUK)
    - •••
- Logic includes decoder for opcode, etc.

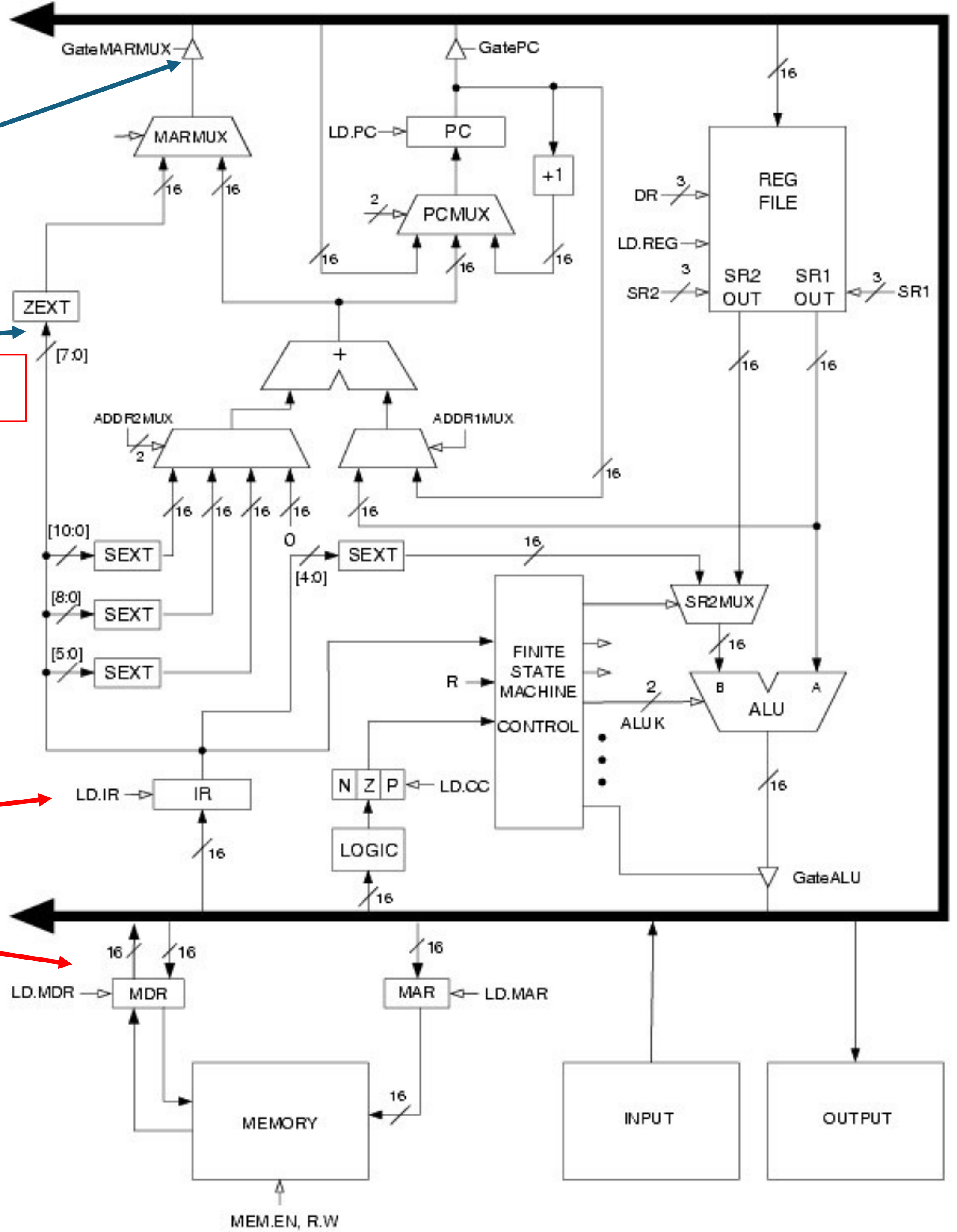


LC-3 Data Path

tri-state devices

Filled arrow = info to be processed

Unfilled arrow = control signal.



Homework

- Exercises:

  5.2, 5.3, 5.4, 5.7, 5.8,

  5.10, 5.13, 5.21, 5.23,

  5.32, 5.35, 5.37, 5.50