

Toward a general purpose computer II

Example: Game of Life

Problems in the previous implementation

1. Similar instructions in different parts of the algorithm require different lines

Very large ROM

Example

$N = N + 1$

$tmp = i + 1$

Target = A + B

Problems in the previous implementation

2. All the variables of the algorithm are stored in registers.

1. Change in the algorithm will require change in hardware
2. Registers are expensive.
3. There is limited space.

Solution to 1 (and 2)

- Define more general instructions
- The algorithm will be a set of the general instructions.

List of instruction in game of life

- $\$3 = \$1 + \$2$
- $\$3 = \$1 - \$2$
- $\$3 = \text{And}(\$1, \$2)$
- $\$3 = \text{OR}(\$1, \$2)$
- $\$3 = \text{Decode}(\$1)$
- $\$3 = \text{set on less } \$1, \$2$
- $\$1 = \text{VAL}$

Register no. 2

The set on less instruction

- $\$3 = \text{set on less } \$1, \$2$
 - $\$3 = 00000\dots01$ if $\$1 < \2
 - $\$3 = 00000\dots00$ otherwise.

ALU codes

The operation	ALU code
$\$3 = \$1 + \$2$	000
$\$3 = \$1 - \$2$	001
$\$3 = \text{And}(\$1, \$2)$	010
$\$3 = \text{OR}(\$1, \$2)$	011
$\$3 = \text{Decode}(\$1)$	100
$\$3 = \text{set on less } \$1, \$2$	101

Solution to 2

- Move the variables to the RAM

We will need instructions to load and store registers in the RAM

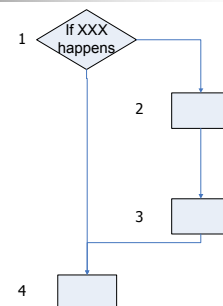
Load and Store instructions

- Load $\text{addr}, \$1$ – load address into $\$1$
- Store $\text{addr}, \$1$ – store $\$1$ into addr

Instruction control

- Goto the next instruction
- j Addr
 - Jump to instruction in address addr

Control the PC branching



Control the PC branching

1. PC = PC+1
2. Check xxx happened
If it didn't PC = 4
3. IR ← PC

```

    graph TD
      1[1. PC = PC+1] --> 2{2. If xxx happens}
      2 -- Yes --> 3[3. IR ← PC]
      2 -- No --> 4[4. ]
      3 --> 4
  
```

PC control

- Jne \$1,\$2,Addr
 - If \$1 ≠ \$2 → Goto Addr
- Je \$1,\$2,Addr
 - If \$1 = \$2 → Goto Addr

The instructions in the system

Arithmetic/Logic operations	Memory related operations	Control operations
<ul style="list-style-type: none"> ■ \$3 = \$1 + \$2 ■ \$3 = \$1 - \$2 ■ \$3 = And(\$1,\$2) ■ \$3 = OR(\$1,\$2) ■ \$3 = Decode(\$1) ■ \$3 = set-on-less(\$1,\$2) ■ \$1 = VAL 	<ul style="list-style-type: none"> ■ Load addr,\$1 ■ Store addr,\$1 	<ul style="list-style-type: none"> ■ j Addr ■ Jne \$1,\$2,Addr ■ Je \$1,\$2,Addr

Instruction

Assumptions:

- We have 16 registers \$1...\$26 (\$0 is always zero).
- The RAM has 65536 entries of 16bits (16bit address)
- The program ROM has 65536 entries of 32bits each.

The structure of instructions

Arithmetic logic

2 bits	3 bits	4 bits	4 bits	4 bits
Type: Arithmetic, memory or control	Sub Type Additions, subtraction	Reg1 address	Reg2 Address	Target Address
[0-4] First 5 bits reserved		[5-8]	[9-12]	[13-16]

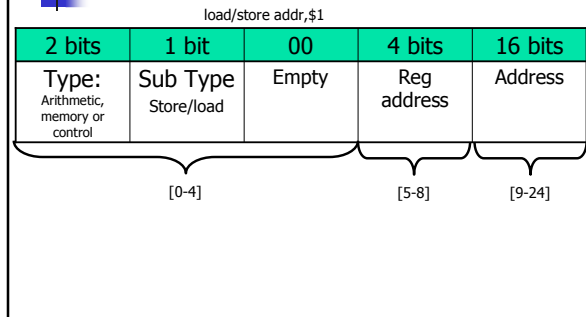
Total 16 registers

The structure of instructions

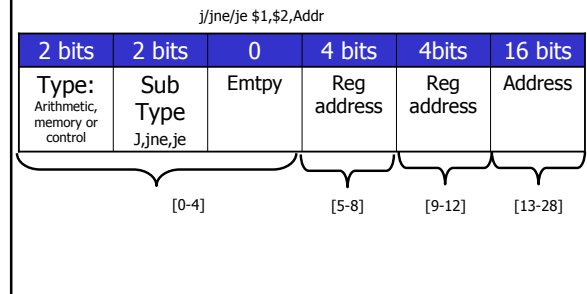
\$1 = VAL

2 bits	111	4 bits	16 bits
Type: Arithmetic, memory or control	Set	Reg1 address	The value
[0-4] First 5 bits reserved		[5-8]	[9-24]

The structure of instructions Memory instruction



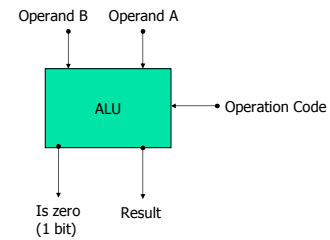
The structure of instructions Control instructions



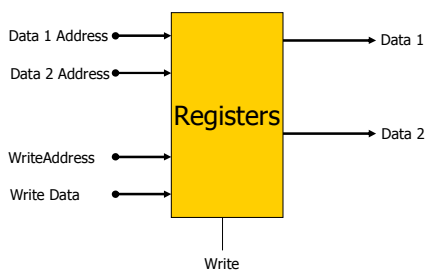
Note about the control instructions

- In the j instruction we ignore the first and second fields.

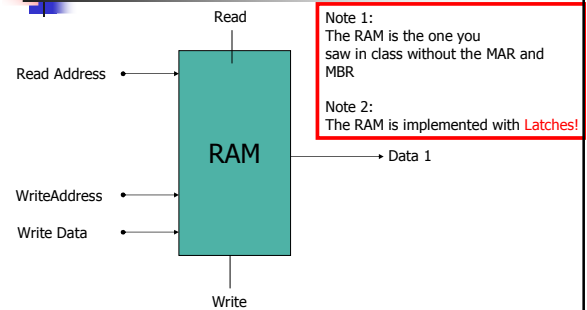
The components of the circuit: The ALU



The components of the circuit: The Registers

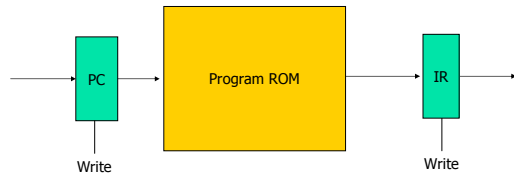


The components of the circuit: The RAM



The components of the circuit: The PC and the program

- PC – holds the next address
- IR - holds the current instruction



A note before implementation

- Several time cycles were lost because not all instructions have the same number of steps.

Solution: Use a counter for the micro-instructions.
CAR

The components of the circuit: CAR, example with arithmetic instruction

Goto the next instruction

CAR = 0: IR ← PC, PC = PC + 1,
CAR = CAR ++

CAR = 1: Perform the code
CAR = 0

The components of the circuit: CAR, example with jne \$1,\$2,Addr

Goto the next instruction

CAR = 0: IR ← PC, PC = PC + 1,
CAR = CAR ++

CAR = 1: \$1 - \$2, if not zero PC ← Addr
CAR = 0

The components of the circuit: CAR, example with arithmetic instruction

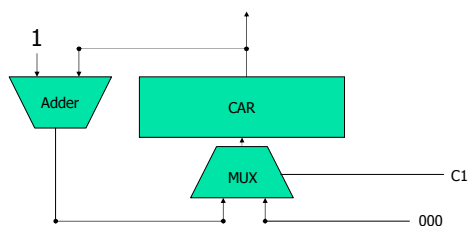
Goto the next instruction

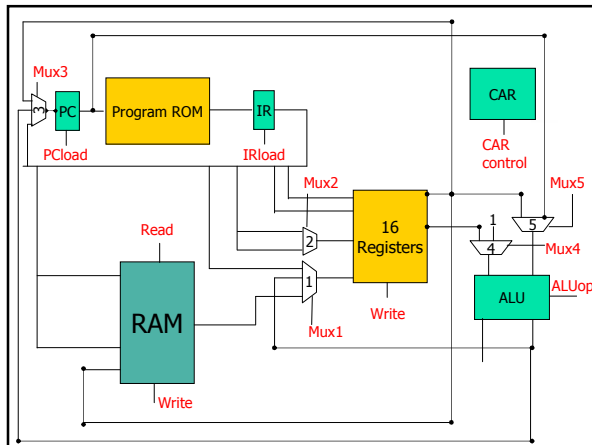
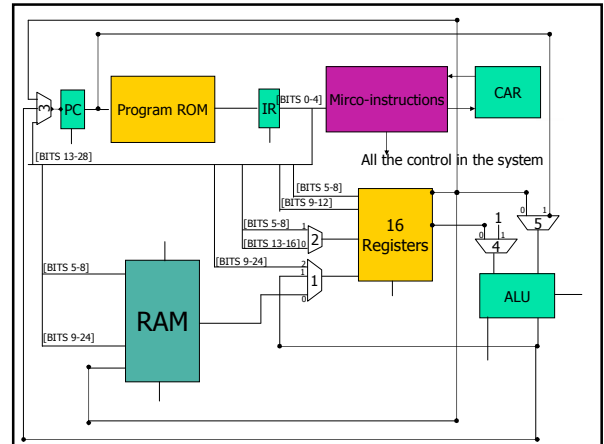
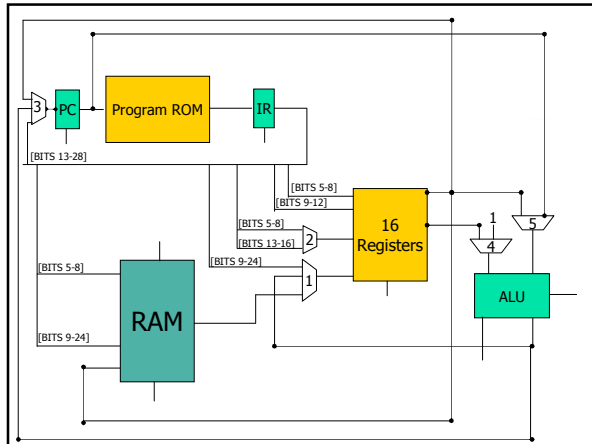
CAR = 0: IR ← PC, PC = PC + 1,
CAR = CAR ++

CAR = 1: Perform the code

For efficiency, we will NOT use the ALU here.

The CAR circuit





The micro-instruction ROM

Inst.	CAR	ALU op	PC load	IR load	RAM read	RAM write	Reg Write	CAR control	Mux					
									1	2	3	4	5	

The micro-instruction ROM Example:

Inst.	CAR	ALUop	PC load	IR load	RAM read	RAM write	Reg Write	CAR control	Mux					
									1	2	3	4	5	
00000	0	000	1	1	0	0	0	1	X	X	1	1	1	1
00000	1	000	0	0	0	0	1	0	2	1	X	0	0	0

CAR = 0: $IR \leftarrow PC$, $PC = PC + 1$,
 $CAR = CAR + +$

CAR = 1: $\$3 = \$2 + \$1$
 $CAR = 0$

The micro-instruction ROM Example:

Inst.	CAR	ALUop	PC load	IR load	RAM read	RAM write	Reg Write	CAR control	Mux					
									1	2	3	4	5	
00000	0	000	1	1	0	0	0	1	X	X	2	1	1	1
00000	1	000	0	0	0	0	1	0	2	1	X	0	0	0

We don't care about these

CAR = 0: $IR \leftarrow PC$, $PC = PC + 1$,
 $CAR = CAR + +$

CAR = 1: $\$3 = \$2 + \$1$
 $CAR = 0$

The meaning is:
 Put in PC the result
 Of PC+1

The micro-instruction ROM

Example:

Inst.	CAR	ALUop	PC load	IR load	RAM read	RAM write	Reg Write	CAR control	Mux				
									1	2	3	4	5
00000	0	000	1	1	0	0	0	1	X	X	2	1	1
00000	1	000	0	0	0	0	1	0	2	1	X	0	0

CAR = 0: $IR \leftarrow PC$, $PC = PC + 1$,
 $CAR = CAR + 1$

CAR = 1: $\$3 = \$2 + \$1$
 $CAR = 0$

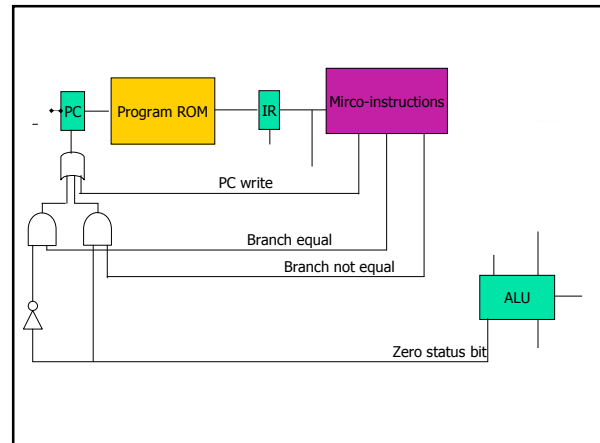
In addition

- In order to implement the jne instruction we need a conditional write on the PC.
- The essence is in here

Addition to the ROM:

Inst.	CAR	Branch Equal	Branch NotEqual

1-bit flags that are used in the jne,je



The program ROM

Address	Instruction	Meaning
0000	0000000100100011	$\$3 = \$2 + \$1$
0001	

- Instruction type 00
- Instruction sub type
- Reg1
- Reg2
- Reg3

Saving more space

- The fetch is divided into 2 cycles
 - Fetch instruction
 - Goto Right instruction

The ROM will depend on the CAR alone

CAR

- Arithmetic = 1 micro instruction
- Load = 2 micro instructions
- Store = 2 micro instructions

CAR – the instruction-CAR table

Instruction	CAR
Arithmetic Operation	00010
Load	00011
	00100
Store	00101
	00110

CAR – Order of execution

Assume we are executing the instruction Load ...

	CAR
Fetch	00000
Goto the right instruction	00011

Instruction	CAR
Arithmetic Operation	00010
Load	00011
	00100
Store	00101
	00110

CAR – Order of execution

Assume we are executing the instruction Load ...

	CAR
Fetch	00000
Goto the right instruction	00011

Instruction	CAR
Arithmetic Operation	00010
Load	00011
	00100
Store	00101
	00110

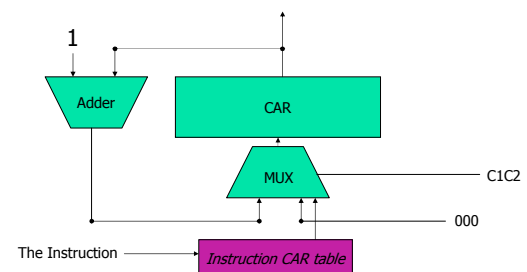
CAR – Order of execution

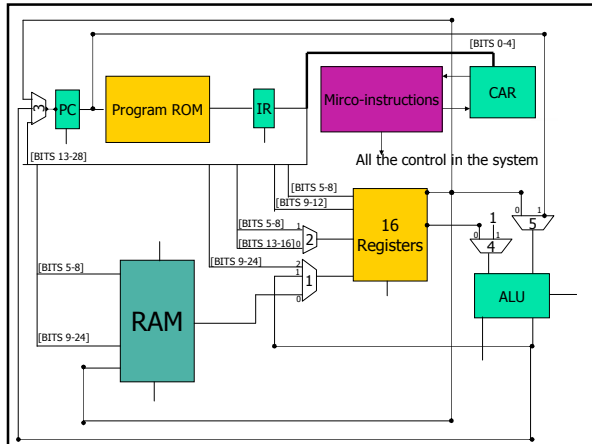
Assume we are executing the instruction Load ...

	CAR
Fetch	00000
Goto the right CAR	00011

Instruction	CAR
Arithmetic Operation	00010
Load	00011
	00100
Store	00101
	00110

Implementation of the Goto in the CAR circuit





The micro-instruction ROM

CAR	ALU op	PC load	IR load	RAM read	RAM write	Reg Write	CAR control	Mux						
								1	2	3	4	5		

The micro instruction ROM depends on the CAR only now.