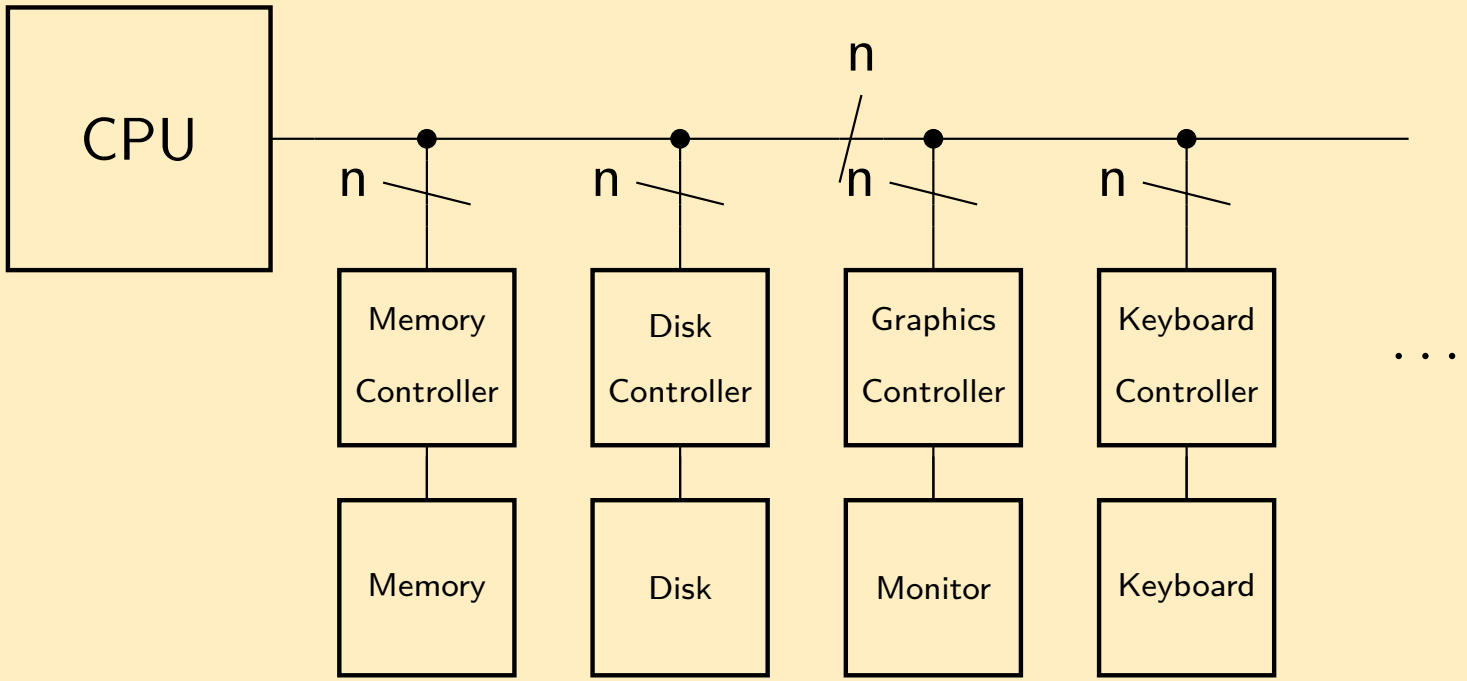
The Carmion

February 3, 2025

The **Carmion-1**

A subset of RISC-V64 was planned, but this will do, I guess

Classical Computer System



- This is how computer systems **used** to look
- **We** can still think they look like this

Processor loop

The **processor** is a **program** in an infinite loop

1. Instruction fetch
2. Instruction decode
3. Instruction execute

The processor consumes **machine instructions**

Carmion's Main Characteristics

- Von Neumann architecture
- Word size is 32 bits
- Main memory size is 1024 words
- The memory is word addressable
- Two registers are available for programming

Carmion's registers

Internal Registers

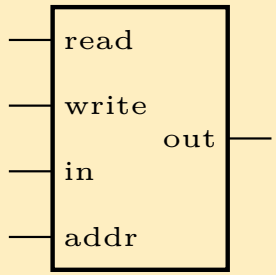
Name	Full name	Width
seq	Sequencer	6
pc	Program Counter	10
ir	Instruction Register	32
ar	Address Register	10
dr	Data Register	32
haltReg	Halt Register	1
ilglReg	Illegal Instruction Register	1

Programmer Accessible

Name	Full name	Width
r0		32
r1		32

Memory (RAM/ROM)

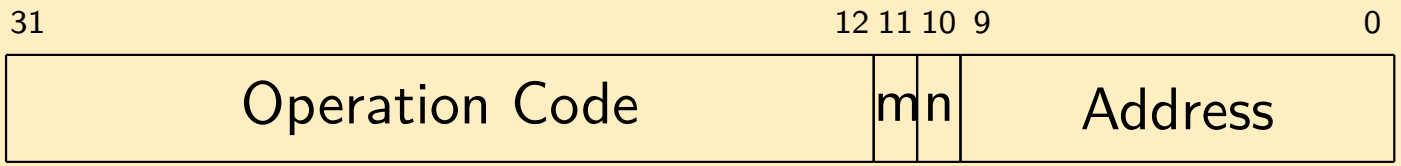
- Memory access nowadays is not exactly trivial
- We will use a simplified memory model
  - Memory responds fast
  - Only word access



Recall: Von Neumann architecture

- Machine instructions are in memory
- The processor needs to bring the instructions in

Carmion's Instructions Format



field	meaning
Operation Code	Identifies the instruction
n	The number of the 'left' register
m	The number of the 'right' register
Address	Memory address of operand

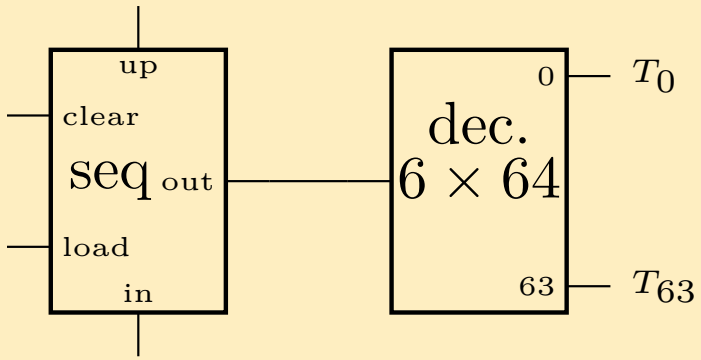
Carmion's Machine Instructions

halt	0000000000000001	xx	xxxxxxxxxx
lw rn,addr	0000000000000010	xn	addr
sw rn,addr	0000000000000011	xn	addr
add rn,rm	0000000000000100	mn	xxxxxxxxxx
bgeu rn,rm	0000000000000101	mn	distance

The Processor RTL

State Machine Conventions

- Each state has a unique number
- The active state number is in the register seq
- No explicit mention of seq on an edge means $\mathrm{seq}++$
- For simplification the value of seq goes into a decoder
- (A peculiar 1-hot method, I guess)



Processor initialization

$$
\begin{array}{ll}
\text{reset}: & \text{seq} \leftarrow 0, \text{r0} \leftarrow 0, \text{r1} \leftarrow 0 \\
T_0: & \text{pc} \leftarrow 512, \\
T_0: & \text{haltReg} \leftarrow 0, \text{ilglReg} \leftarrow 0 \\
T_0: & \text{ir} \leftarrow 0
\end{array}
$$

The reset on $r[0]$ and $r[1]$ is a kludge for the alu flags

Instruction Fetch

$$
\begin{aligned}
T_1 &: \quad \mathrm{ar} \leftarrow \mathrm{pc}, \mathrm{pc} \leftarrow \mathrm{pc} + 1 \\
T_2 &: \quad \mathrm{dr} \leftarrow m[\mathrm{ar}] \\
T_3 &: \quad \mathrm{ir} \leftarrow \mathrm{dr}
\end{aligned}
$$

Instruction Decode

$$
\begin{array}{ll}
 & \mathrm{op} = \mathrm{ir}[31:12] \\
T_4(\mathrm{op} \neq 0)(\mathrm{op} < 6): & \mathrm{seq} \leftarrow \mathrm{op} << 3 \\
T_5: & \mathrm{haltReg} \leftarrow \mathrm{haltReg} + 1 \\
T_6: & \mathrm{ilglReg} \leftarrow \mathrm{ilglReg} + 1 \\
T_7: & \mathrm{seq} \leftarrow 7
\end{array}
$$

halt (**halt**)

Processor stops

$$T_8: \quad \text{haltReg} \leftarrow \text{haltReg} + 1$$
$$T_9: \quad \text{seq} \leftarrow 9$$

lw rn,addr (**L**oad **w**ord)

$r[n] \leftarrow [\text{addr}]$

$$
\begin{aligned}
T_{16}: &\quad \text{ar} \leftarrow \text{ir}[9:0] \\
T_{17}: &\quad \text{dr} \leftarrow m[\text{ar}] \\
T_{18}: &\quad r[\text{ir}[10]] \leftarrow \text{dr}, \text{seq} \leftarrow 0
\end{aligned}
$$

sw rn,addr (**s**tore **w**ord)

$[\text{addr}] \leftarrow r[n]$

$$
\begin{aligned}
T_{24}: &\quad \text{dr} \leftarrow r[\text{ir}[10]], \text{ar} \leftarrow \text{ir}[9:0] \\
T_{25}: &\quad m[\text{ar}] \leftarrow \text{dr}, \text{seq} \leftarrow 0
\end{aligned}
$$

add rn,rm (**add**)

$r[n] \leftarrow r[n] + r[m]$

$$T_{32}: \quad r[\mathrm{ir}[10]] \leftarrow r[\mathrm{ir}[10]] + r[\mathrm{ir}[11]], \mathrm{seq} \leftarrow 0$$

bgeu rn,rm,off (**b**ranch **g**reater **e**qual) **u**nsigned

If $r[n] \geq_u r[m]$ then $\mathrm{pc} \leftarrow \mathrm{pc} + \mathrm{ir}[9:0]$

$$T_{40}(r[\mathrm{ir}[10]] \geq_u r[\mathrm{ir}[11]]): \quad \mathrm{pc} \leftarrow \mathrm{pc} + \mathrm{ir}[9:0]$$
$$T_{40}: \quad \mathrm{seq} \leftarrow 0$$

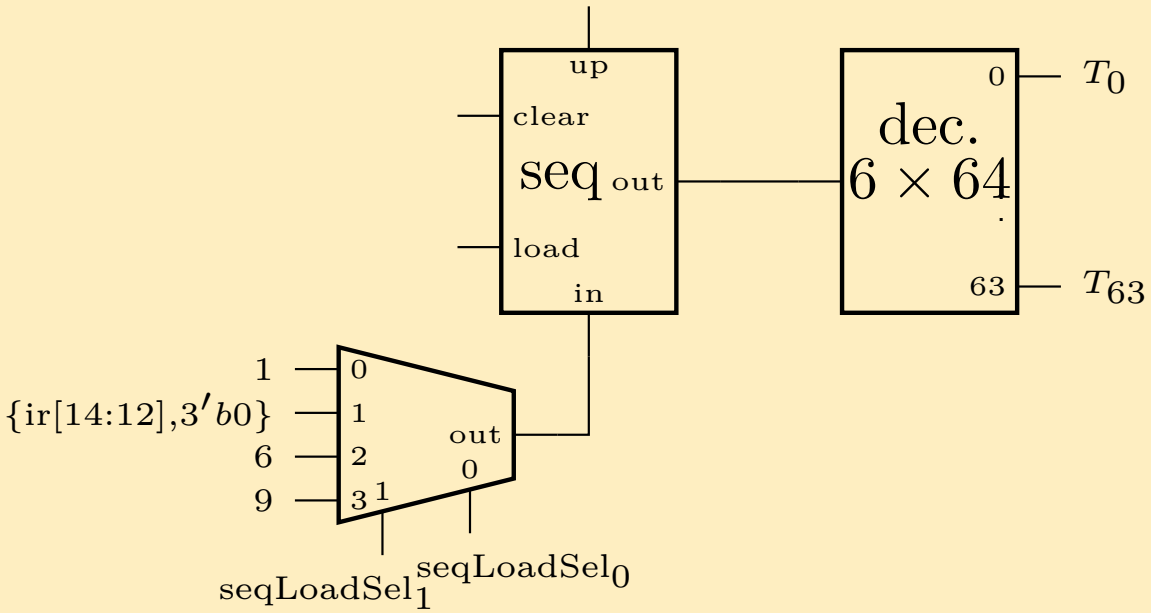
‘Building’ the Hardware

The seq register

seq to the left of $\leftarrow$ (RTL)

$$
\begin{array}{ll}
\text{reset}: & \text{seq} \leftarrow 0 \\
T_4(0 \neq \text{op} < 6): & \text{seq} \leftarrow \text{op} << 3 \\
T_6: & \text{seq} \leftarrow 6 \\
T_9: & \text{seq} \leftarrow 9 \\
T_{18}: & \text{seq} \leftarrow 0 \\
T_{25}: & \text{seq} \leftarrow 0 \\
T_{32}: & \text{seq} \leftarrow 0 \\
T_{40}: & \text{seq} \leftarrow 0
\end{array}
$$

seq to the left of ← (data path)



seq to the left of ← (formulae)

$$\text{seqClear} = \text{reset}$$

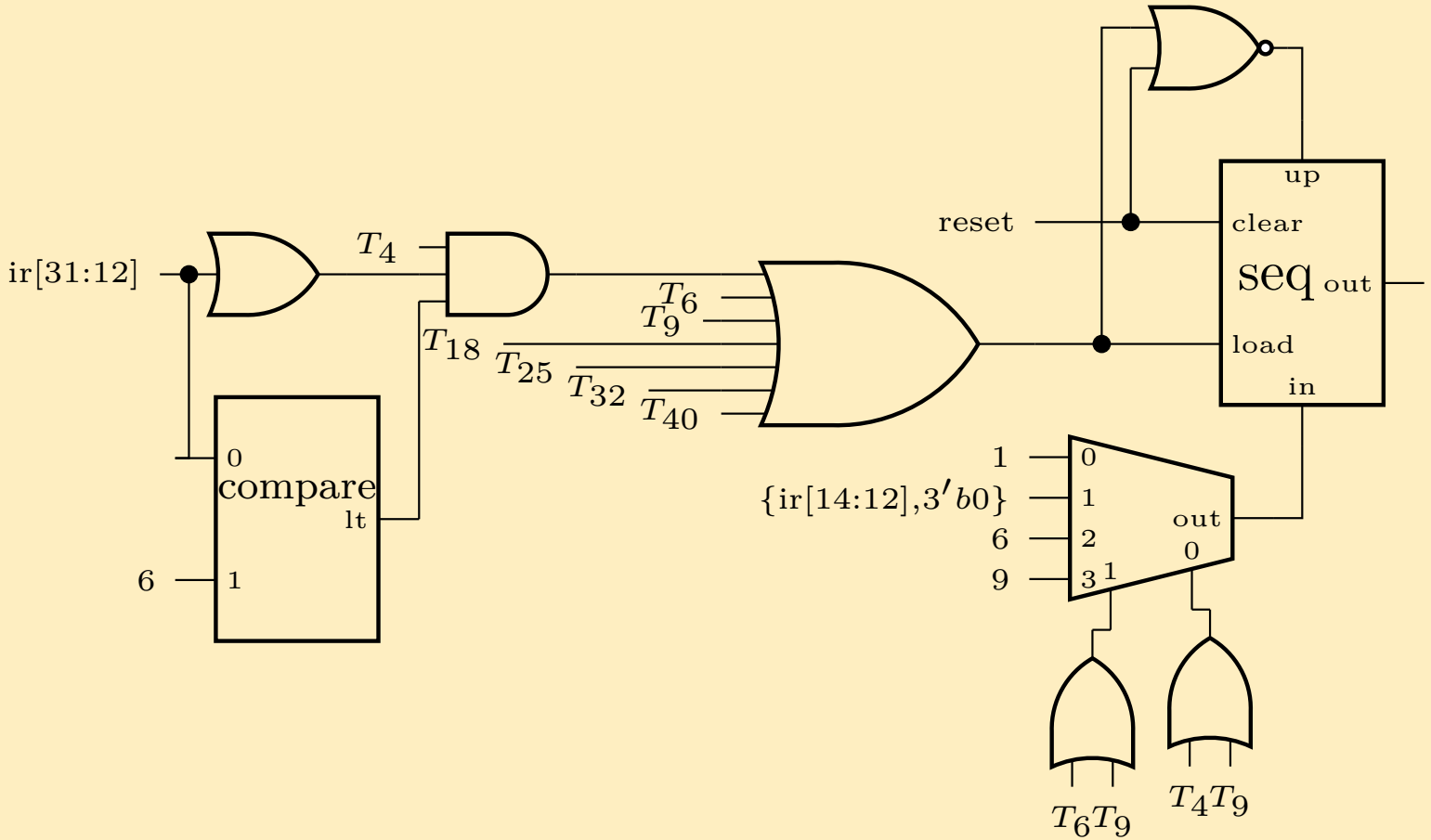
$$t = T_{18} + T_{25} + T_{32} + T_{40}$$

$$\text{seqLoad} = T_4(\text{op} < 6)(\text{ir}[31] + \cdots + \text{ir}[12]) + T_6 + T_9 + t$$

$$\text{seqLoadSel}_1 = T_6 + T_9$$

$$\text{seqLoadSel}_0 = T_4 + T_9$$

seq to the left of ← (control)

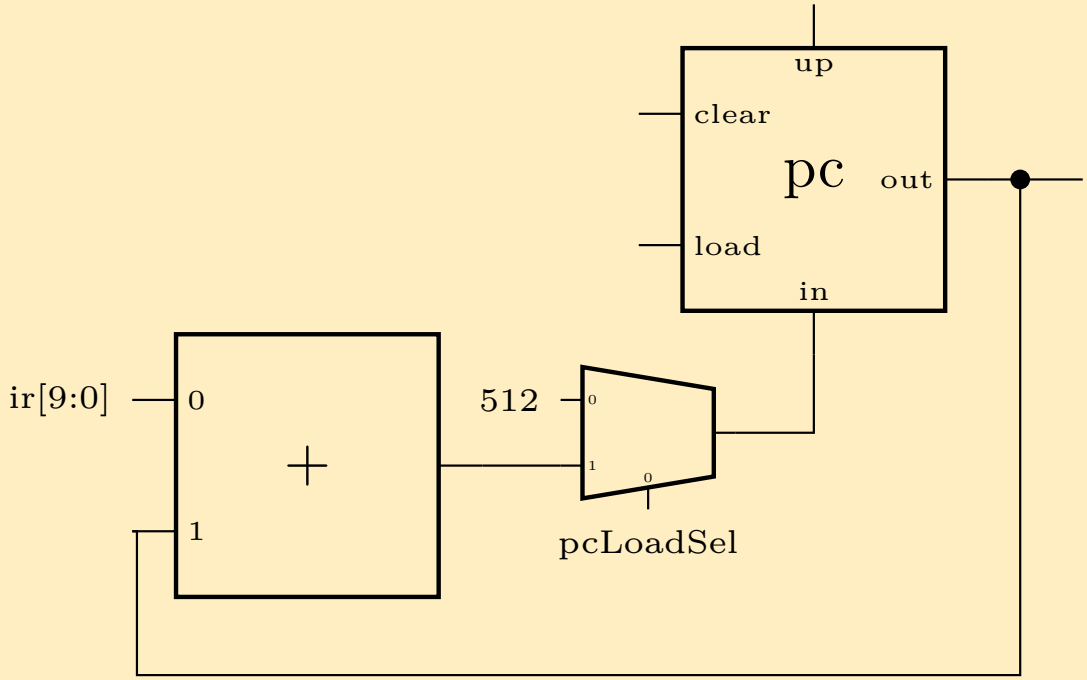


The pc register

pc to the left of $\leftarrow$ (RTL)

$$
\begin{array}{ll}
\text{reset}: & \text{pc} \leftarrow 512 \\
T_1: & \text{pc} \leftarrow \text{pc} + 1 \\
T_{40}(r[\text{ir}[10]] \geq_u r[\text{ir}[11]]): & \text{pc} \leftarrow \text{pc} + \text{ir}[9:0]
\end{array}
$$

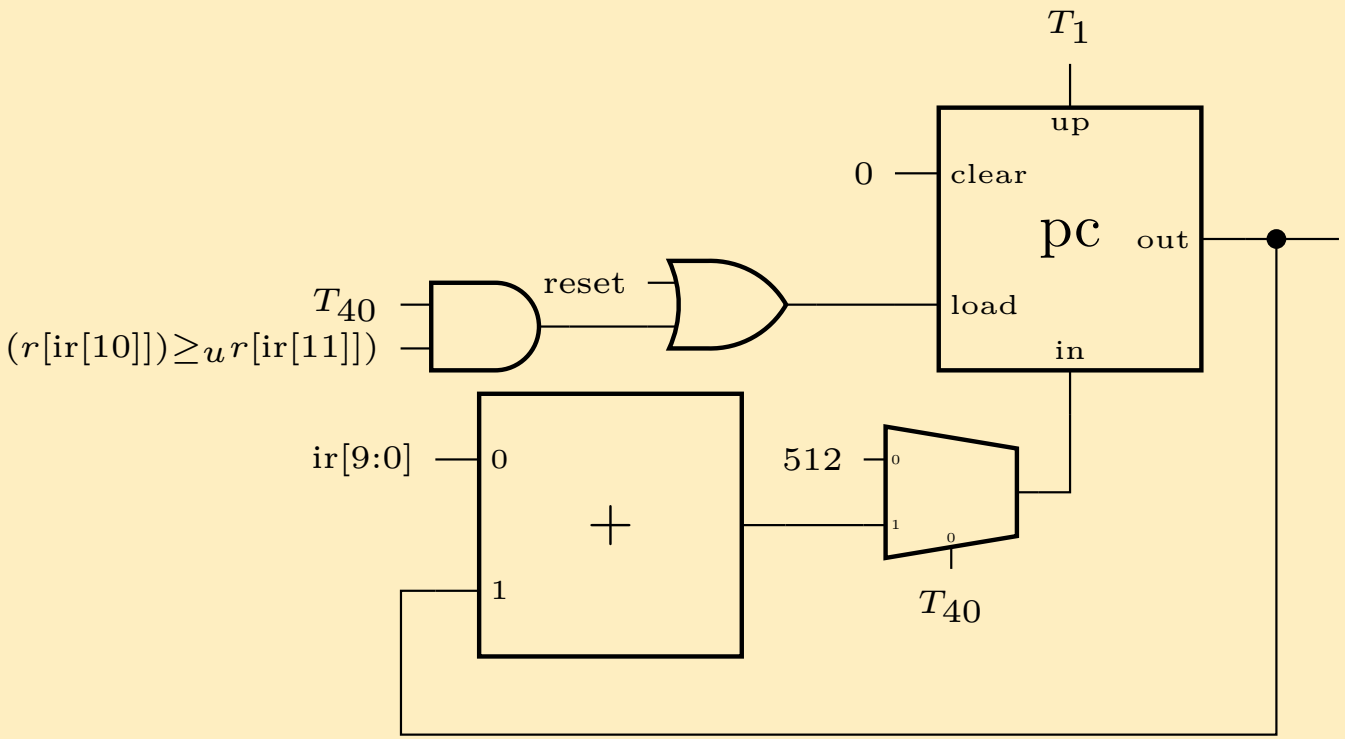
pc to the left of ← (data path)



pc to the left of ← (formulae)

$$\text{pcLoad} = \text{reset} + T_{40}(r[\text{ir}[10]] \geq_u \text{ir}[11]])$$
$$\text{pcLoadSel} = T_{40}$$
$$\text{pcUp} = T_1$$

pc to the left of ← (control)



The haltReg register

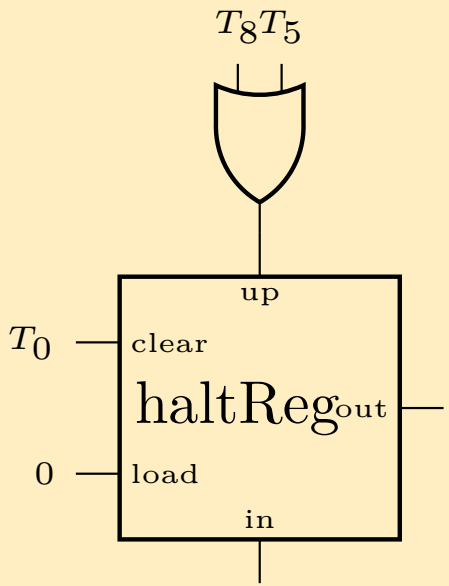
haltReg to the left of $\leftarrow$ (RTL)

$$
\begin{aligned}
T_0: &\quad \text{haltReg} \leftarrow 0 \\
T_5: &\quad \text{haltReg} \leftarrow \text{haltReg} + 1 \\
T_8: &\quad \text{haltReg} \leftarrow \text{haltReg} + 1
\end{aligned}
$$

haltReg to the left of $\leftarrow$ (formulae)

$$\text{haltRegClear} = T_0$$
$$\text{haltRegUp} = T_5 + T_8$$

haltReg to the left of ← (control)



The ilglReg register

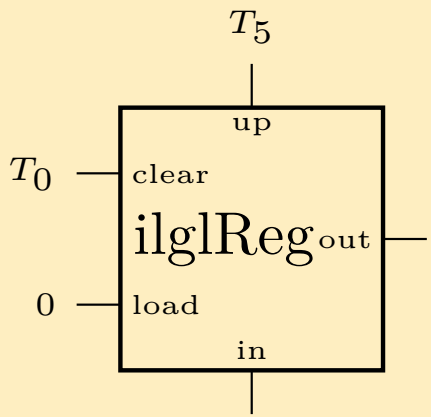
ilglReg to the left of $\leftarrow$ (RTL)

$$T_0: \quad \text{ilglReg} \leftarrow 0$$
$$T_5: \quad \text{ilglReg} \leftarrow \text{ilglReg} + 1$$

ilglReg to the left of $\leftarrow$ (formulae)

$$\text{ilglRegClear} = T_0$$
$$\text{ilglRegUp} = T_5$$

ilglReg to the left of ← (control)

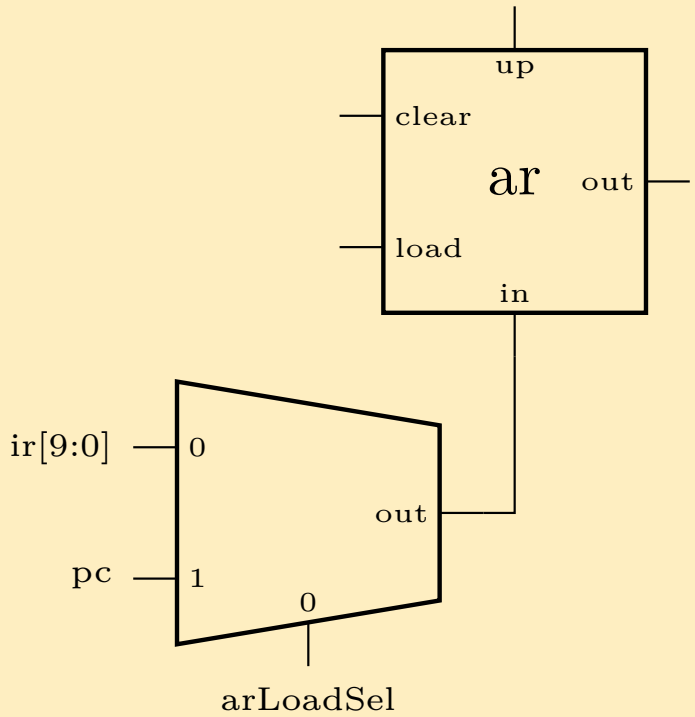


The ar register

ar to the left of $\leftarrow$ (RTL)

$$
\begin{aligned}
T_1 :&\quad \mathrm{ar} \leftarrow \mathrm{pc} \\
T_{16} :&\quad \mathrm{ar} \leftarrow \mathrm{ir}[9:0] \\
T_{24} :&\quad \mathrm{ar} \leftarrow \mathrm{ir}[9:0]
\end{aligned}
$$

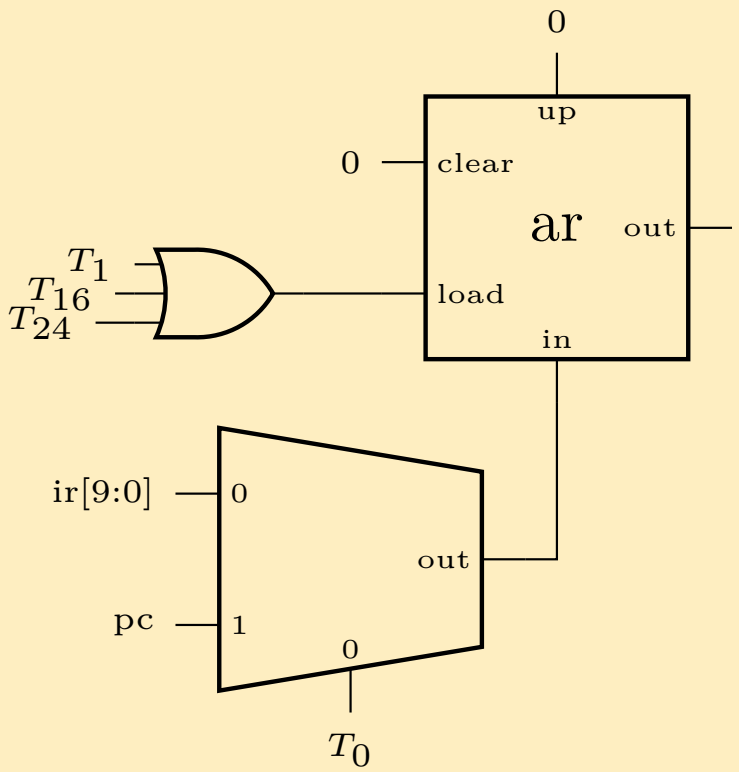
ar to the left of ← (data path)



ar to the left of $\leftarrow$ (formulae)

$$\text{arLoad} = T_1 + T_{16} + T_{24}$$
$$\text{arLoadSel} = T_0$$

ar to the left of ← (circuit)

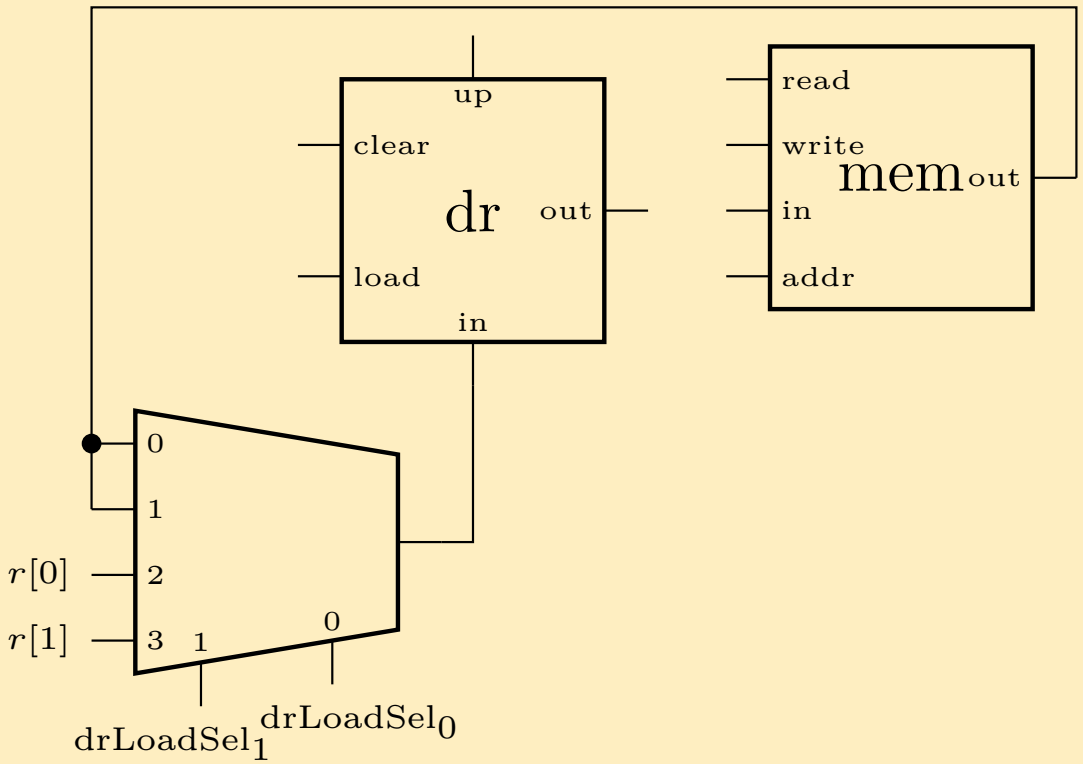


The dr register

dr to the left of $\leftarrow$ (RTL)

$$
\begin{aligned}
T_2 :&\quad \mathrm{dr} \leftarrow m[\mathrm{ar}] \\
T_{17} :&\quad \mathrm{dr} \leftarrow m[\mathrm{ar}] \\
T_{24} :&\quad \mathrm{dr} \leftarrow r[\mathrm{ir}[10]]
\end{aligned}
$$

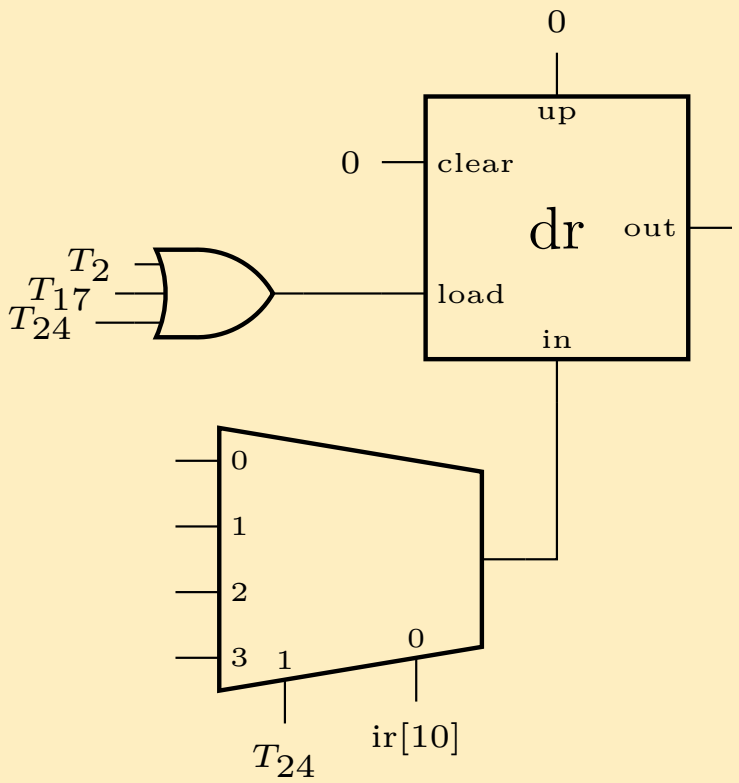
dr to the left of ← (data)



dr to the left of ← (formulae)

$$\mathrm{drLoad} = T_2 + T_{17} + T_{24}$$
$$\mathrm{drLoadSel}_1 = T_{24}$$
$$\mathrm{drLoadSel}_0 = \mathrm{ir}[10]$$

dr to the left of ← (control)

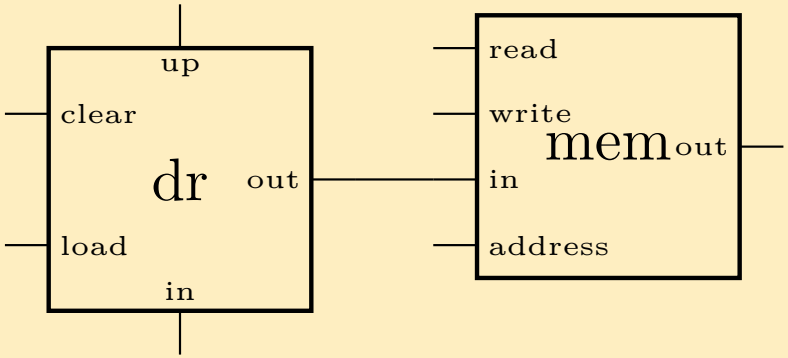


Memory Access

Memory Access (RTL)

$$
\begin{aligned}
T_2 : &\quad \mathrm{dr} \leftarrow m[\mathrm{ar}] \\
T_{17} : &\quad \mathrm{dr} \leftarrow m[\mathrm{ar}] \\
T_{25} : &\quad m[\mathrm{ar}] \leftarrow \mathrm{dr}
\end{aligned}
$$

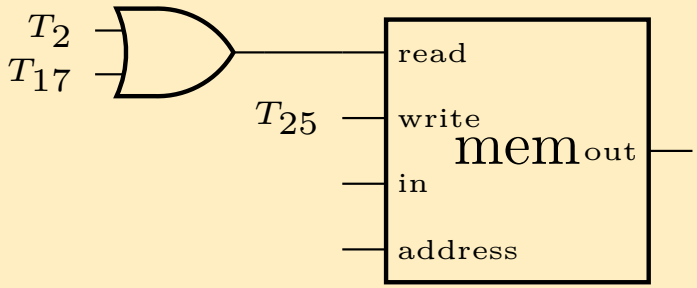
Memory Access (data path)



Memory access (formulae)

$$\text{memRead} = T_2 + T_{17}$$
$$\text{memWrite} = T_{25}$$

Memory Access (control)

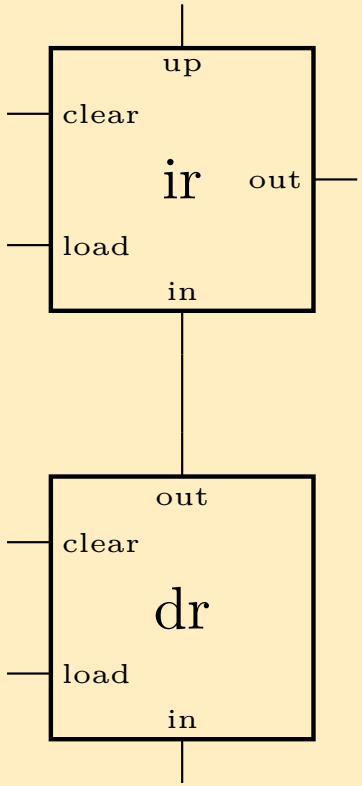


The ir register

ir to the left of $\leftarrow$ (RTL)

$$T_3: \quad \mathrm{ir} \leftarrow \mathrm{dr}$$

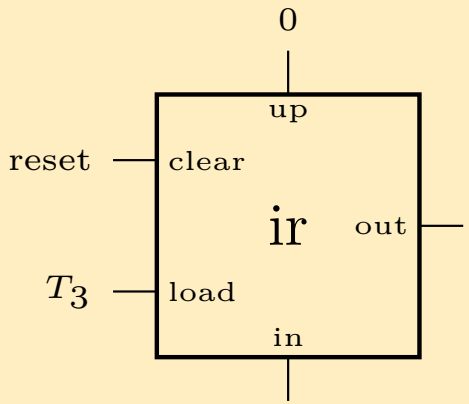
ir to the left of ← (control)



ir to the left of $\leftarrow$ (formulae)

$$\text{irClear} = \text{reset}$$
$$\text{irLoad} = T_3$$

ir to the left of ← (control)

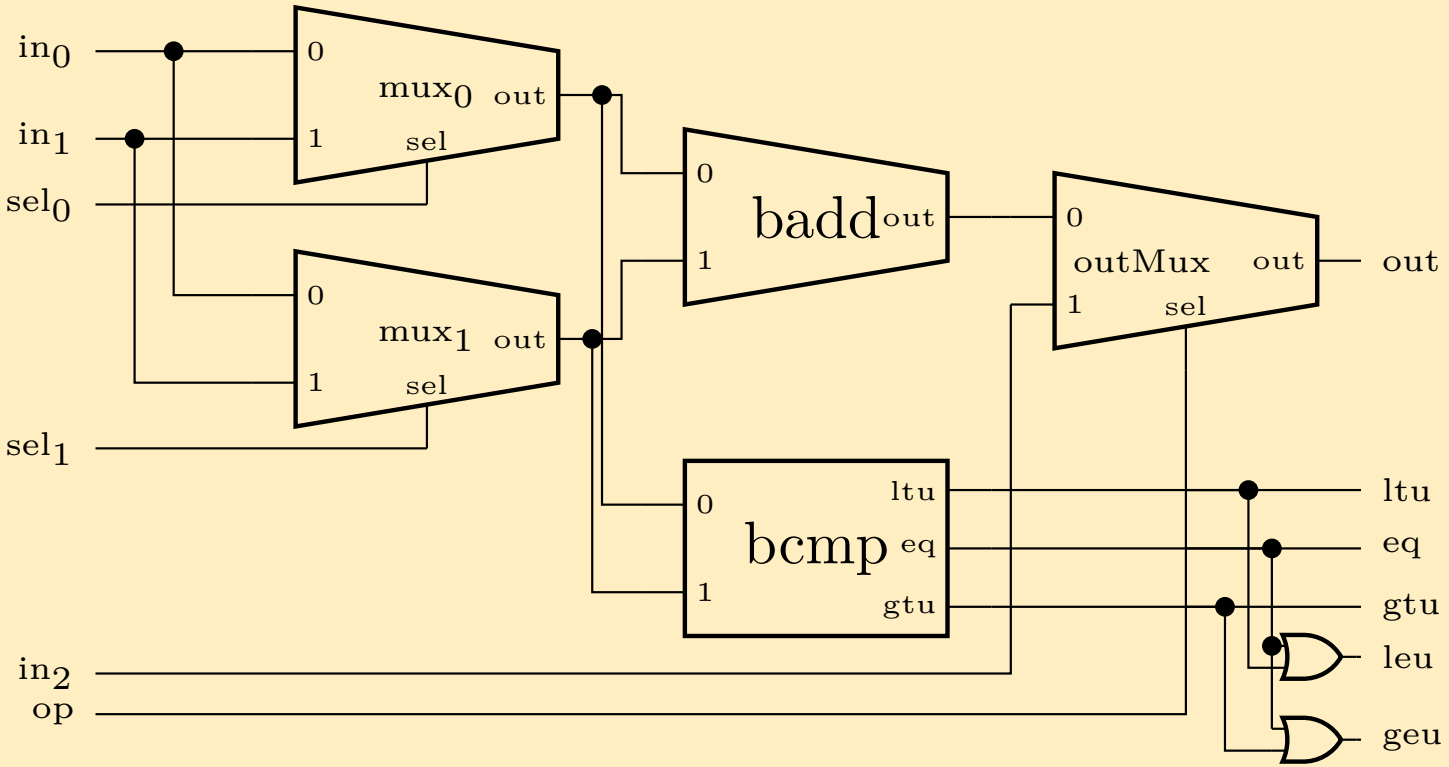


$r[n]$ to the left of $\leftarrow$

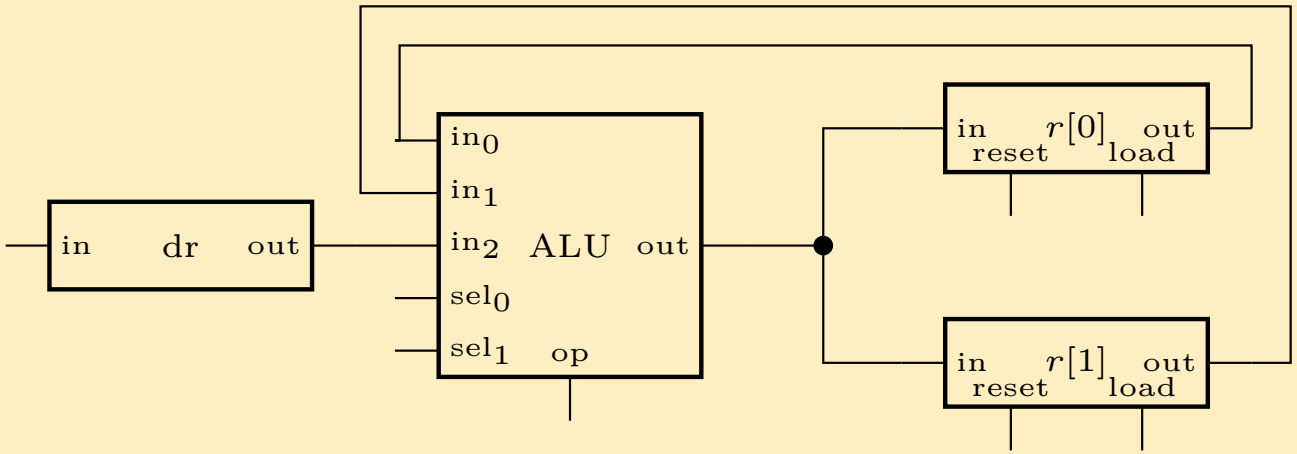
$r[n]$ to the left of $\leftarrow$ (RTL)

$$
\begin{array}{ll}
\text{reset}: & r[0] \leftarrow 0, r[1] \leftarrow 0 \\
T_{18}: & r[\text{ir}[10]] \leftarrow \text{dr} \\
T_{32}: & r[\text{ir}[10]] \leftarrow r[\text{ir}[10]] + r[\text{ir}[11]]
\end{array}
$$

ALU (data path)



$r[n]$ to the left of $\leftarrow$ (data path)



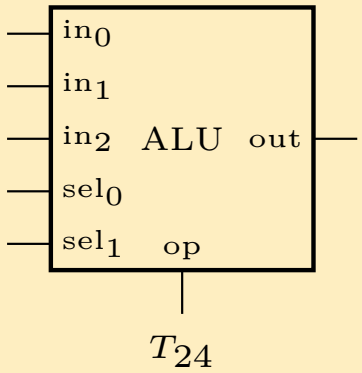
$r[n]$ to the left of $\leftarrow$ (formulae)

$$\text{r0Reset} = \text{reset}$$
$$\text{r0Load} = T_{18}\overline{\text{ir}[10]} + T_{32}\overline{\text{ir}[10]}$$

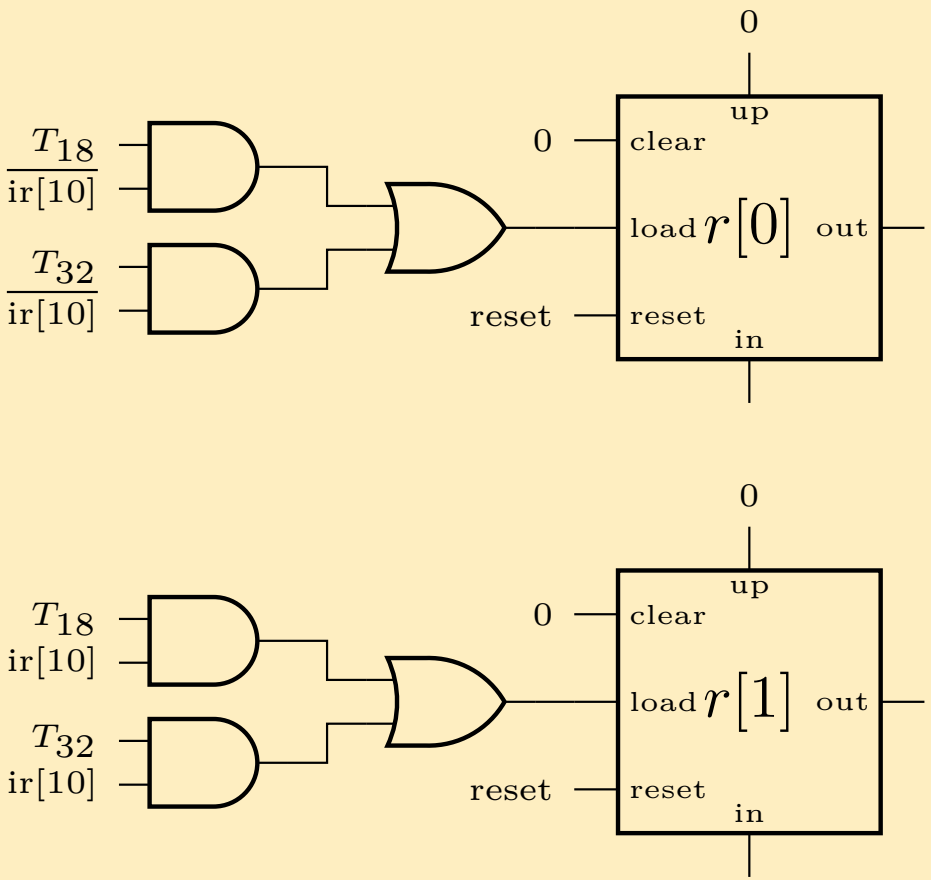
$$\text{r1Reset} = \text{reset}$$
$$\text{r1Load} = T_{18}\text{ir}[10] + T_{32}\text{ir}[10]$$

$$\text{aluSel}_0 = \text{ir}[10]$$
$$\text{aluSel}_1 = \text{ir}[11]$$
$$\text{aluOp} = T_{24}$$

ALU (control)



$r[n]$ to the left of $\leftarrow$ (control)



carmion1 datapath I

Listing 1: demo/01_carmion1/dp.v

```
`timescale 1ns/1ns

module dp(
    output aluEq,aluGeu,aluGtu,aluLeu,aluLtu,
    output [9:0]  arOut,
    output [31:0] drOut,
    output [31:0] irOut,
    input  aluOp,
    input  arLoad, arLoadSel,
    input  [31:0]drInOpt,
    input  drLoad,
    input  [1:0]drLoadSel,
    input  irClear, irLoad,
```

carmion1 datapath II

```
    input  pcLoad, pcLoadSel, pcUp,
    input  r0Load,
    input  r1Load,
    input  reset,
    input  clk
);
    //
    // pc
    //
    wire [9:0]pcOut,pcIn;
    register #(10) pc(
        pcOut,pcIn,,
        1'b0, 1'b0,
        1'b0,
        pcUp, pcLoad, 1'b0,
```

carmion1 datapath III


```
        clk
    );

    wire [9:0]pcOptions[2];
    wire [9:0]pcAdd;
    assign pcOptions[0] = 10'(512);
    assign pcOptions[1] = pcAdd;
    binaryAdder #(10) c_pcadd(,,pcAdd,pcOut,irOut[9
    muxMulti #(1,10) cpcIn(pcIn,pcOptions,pcLoadSel

    //
    // ar
    //
    wire [9:0]arIn;
    register #(10) ar(
```

carmion1 datapath IV


```
        arOut,arIn,,
        1'b0, 1'b0,
        1'b0, 1'b0,
        arLoad, 1'b0,
        clk
    );
    wire [9:0]arOptions[2];
    assign arOptions[0] = irOut[9:0];
    assign arOptions[1] = pcOut;
    muxMulti #(1,10) c_arIn(arIn, arOptions, arLoad


    //
    // dr
    //
```

carmion1 datapath V

```
    wire [31:0] drIn;
    wire drLoad;
    register #(32) dr(
        drOut,drIn,,
        1'b0, 1'b0,
        1'b0, 1'b0,
        drLoad, 1'b0,
        clk
    );
    wire [31:0] drOptions[4];
    assign drOptions[0] = drInOpt;
    assign drOptions[1] = drInOpt;
    assign drOptions[2] = r0Out;
    assign drOptions[3] = r1Out;
    muxMulti #(2,32) c_drIn(drIn, drOptions, drLoad
```

carmion1 datapath VI

```
    //
    // ir
    //
    register #(32) ir(
        irOut,drOut,,
        1'b0, 1'b0,
        1'b0, 1'b0,
        irLoad, irClear,
        clk
    );


    //
    // alu
```

carmion1 datapath VII

```
    //

    wire [31:0] r1Out,r0Out;
    register #(32) r0(
        r0Out,aluOut,,
        1'b0, 1'b0,
        1'b0, 1'b0,
        r0Load, reset,
        clk
    );
    register #(32) r1(
        r1Out,aluOut,,
        1'b0, 1'b0,
        1'b0, 1'b0,
        r1Load, reset,
```

carmion1 datapath VIII


```
        clk
    );

    wire [31:0] aluOut;
    wire aluSel0,aluSel1;
    wire aluLtu,aluEq,aluGtu,aluLeu,aluGeu;
    alu alu(aluEq,aluGeu,aluGtu,aluLeu,aluLtu,
            aluOut,
            r0Out,r1Out,drOut,
            irOut[10],irOut[11],aluOp);

endmodule
```

carmion1 alu (datapath) I

Listing 2: demo/01_carmion1/alu.v

```
`timescale 1ns/1ns

//
// op:
//    2 - dr
//
module alu(
    output eq,geu,gtu,leu,ltu,
    output [31:0]out,
    input  [31:0]in0,in1,in2,
    input  sel0, sel1,
    input  op
);
```

carmion1 alu (datapath) II


```
wire [31:0]opt0[2], opt1[2];
wire [31:0]arg0,arg1;
assign opt0[0] = in0;
assign opt0[1] = in1;
assign opt1[0] = in0;
assign opt1[1] = in1;
muxMulti #(1,32) c_arg0(arg0,opt0,sel0);
muxMulti #(1,32) c_arg1(arg1,opt1,sel1);

wire [31:0]sum;
binaryAdder #(32) ba(,,sum,arg0,arg1,1'b0);

wire [31:0]options[2];
```

carmion1 alu (datapath) III


```
    assign options[0] = sum;
    assign options[1] = in2;

    muxMulti #(1,32) c_out(out,options,op);

    comparator #(32) cond(ltu,eq,gtu,arg0,arg1);

    orGate c_aluLequ(leu,ltu,eq);
    orGate c_aluGequ(geu,gtu,eq);

endmodule
```

carmion1 datapath I

Listing 3: demo/01_carmion1/ctl.v

```
‘timescale 1ns/1ns

parameter opCodeHalt = 1;
parameter opCodeLw   = 2;
parameter opCodeSw   = 3;
parameter opCodeAdd  = 4;
parameter opCodeBgeu = 5;

parameter T_SHIFT = 3;
parameter T_OP    = (2**T_SHIFT);

parameter T_FETCH = 1;
parameter T_SWITCH = 4;
```

carmion1 datapath II

```
parameter T_ilgl = 5;
parameter T_ilglDone = T_ilgl + 1;

parameter T_halt = opCodeHalt * T_OP;
parameter T_haltDone = T_halt + 1;


parameter T_lw      = opCodeLw * T_OP;
parameter T_lwDone  = T_lw + 2;

parameter T_sw      = opCodeSw * T_OP;
parameter T_swDone = T_sw + 1;

parameter T_add     = opCodeAdd * T_OP;
```

carmion1 datapath III

```
parameter T_addDone = T_add;

parameter T_bgeu     = opCodeBgeu * T_OP;
parameter T_bgeuDone = T_bgeu;

module ctl(
    output aluOp,
    output arLoad, arLoadSel,
    output drLoad,
    output [1:0]drLoadSel,
    output flagHalt, flagIlgl,
    output irClear, irLoad,
    output memRead, memWrite,
    output pcLoad, pcLoadSel, pcUp,
    output r0Load,
```

carmion1 datapath IV

```
    output r1Load,
    input  aluEq, aluGeu, aluGtu, aluLeu, aluLtu,
    input  [31:0]ir,
    input  reset,
    input  clk
);
    //
    // seq
    //
    wire [(T_SHIFT+3)-1:0]seqOut,seqIn;
    wire seqClear,seqLoad,seqUp;
    register #(T_SHIFT+3) seq(
        seqOut,seqIn,,
        1'b0, 1'b0,
        1'b0,
```

carmion1 datapath V


```
        seqUp, seqLoad, seqClear,
        clk
    );

    wire [2**(T_SHIFT+3)-1:0]t;
    decoder #(T_SHIFT+3) timing(t,seqOut);

    wire [1:0]seqLoadSel;
    wire [T_SHIFT+3-1:0]seqOptions[4];
    assign seqOptions[0] = T_FETCH;
    assign seqOptions[1] = {ir[14:12],{(T_SHIFT)'(0
    assign seqOptions[2] = T_ilglDone;
    assign seqOptions[3] = T_haltDone;
    orGate c_seqLoadSel1(seqLoadSel[1],t[T_ilglDone
    orGate c_seqLoadSel0(seqLoadSel[0],t[T_SWITCH],
```

carmion1 datapath VI


```
    muxMulti #(2,T_SHIFT+3) cseqIn(seqIn,seqOptions

    wire opLegal;
    comparator #(20) c_opLegal(opLegal,,,ir[31:12],
    andGate c_seqSwitch(seqSwitch,t[T_SWITCH],opLeg

    wire [5:0]allDone;
    assign allDone[0]=seqSwitch;
    assign allDone[1]=t[T_haltDone];
    assign allDone[2]=t[T_lwDone];
    assign allDone[3]=t[T_swDone];
    assign allDone[4]=t[T_addDone];
    assign allDone[5]=t[T_bgeuDone];

    orMulti #(6) c_seqLoad(seqLoad,allDone);
```

carmion1 datapath VII


```
orGate cseqClear(seqClear,reset,reset);

norGate cseqUp(seqUp,seqLoad,seqClear);


//
// pc
//
andGate c_condGeu(condGeu,t[T_bgeu],aluGeu);
orGate c_pcLoad(pcLoad,t[0],condGeu);
assign pcLoadSel = t[T_bgeu];
assign pcUp=t[1];
```

carmion1 datapath VIII


```
    //
    // haltReg
    //
    wire haltOut,haltUp;
    register #(1) chaltReg(
        haltOut,,,
        1'b0, 1'b0,
        1'b0, haltUp,
        1'b0, t[0],
        clk
    );
    orGate chaltUp(haltUp,t[T_ilgl],t[T_halt]);
    assign flagHalt=haltOut;

    //
```

carmion1 datapath IX


```
    // ilglReg
    //
    wire ilglOut,ilglUp;
    register #(1) cilglReg(
        ilglOut,,,
        1'b0, 1'b0,
        1'b0,
        ilglUp, 1'b0, t[0],
        clk
    );
    assign ilglUp = t[T_ilgl];
    assign flagIlgl = ilglOut;


    //
```

carmion1 datapath X


```
    // ar
    //
    or3Gate carLoad(arLoad,t[1],t[T_lw],t[T_sw]);
    assign arLoadSel = t[1];


    //
    // dr
    //
    or3Gate cdrLoad(drLoad,t[2],t[T_lw+1],t[T_sw]);
    assign drLoadSel={{t[T_sw]},{ir[10]}};


    //
    // Memory
```

carmion1 datapath XI


```
    //
    orGate cmemRead(memRead,t[2],t[T_lw+1]);
    assign memWrite=t[T_sw+1];


    //
    // ir
    //
    assign irLoad=t[3];
    assign irClear=reset;


    //
    // r[n]
    //
```

carmion1 datapath XII


```
    assign aluOp = t[T_lw+2];

    notGate ir10n(ir10n,ir[10]);
    andGate cr0loadLw (r0LoadLw ,t[T_lw+2] ,ir10n);
    andGate cr0loadAdd(r0LoadAdd,t[T_add],ir10n);
    orGate  c_r0Load(r0Load,r0LoadLw,r0LoadAdd);

    andGate cr1loadLw (r1LoadLw, t[T_lw+2] ,ir[10]);
    andGate cr1loadAdd(r1LoadAdd,t[T_add],ir[10]);
    orGate  c_r1Load(r1Load,r1LoadLw,r1LoadAdd);
endmodule
```

Finally, **Software**

$\sum_{i=0}^{n} i$ Software

			.code	
512	010000000000000	loop:	lw	r0,sum
513	010010000000001		lw	r1,i
514	100100000000000		add	r0,r1
515	011000000000000		sw	r0,sum
516	010000000000011		lw	r0,one
517	100010000000000		add	r1,r0
518	011010000000001		sw	r1,i
519	010000000000010		lw	r0,n
520	101101111110111		bgeu	r0,r1,loop
521	001000000000000		halt	
			.data	
0	000000000000000	sum:	.word	0
1	000000000000000	i:	.word	0
2	000000000001010	n:	.word	10
3	000000000000001	one:	.word	1

Assembler Symbol Table

Label	Address
i	1
n	2
loop	512
one	3
sum	0