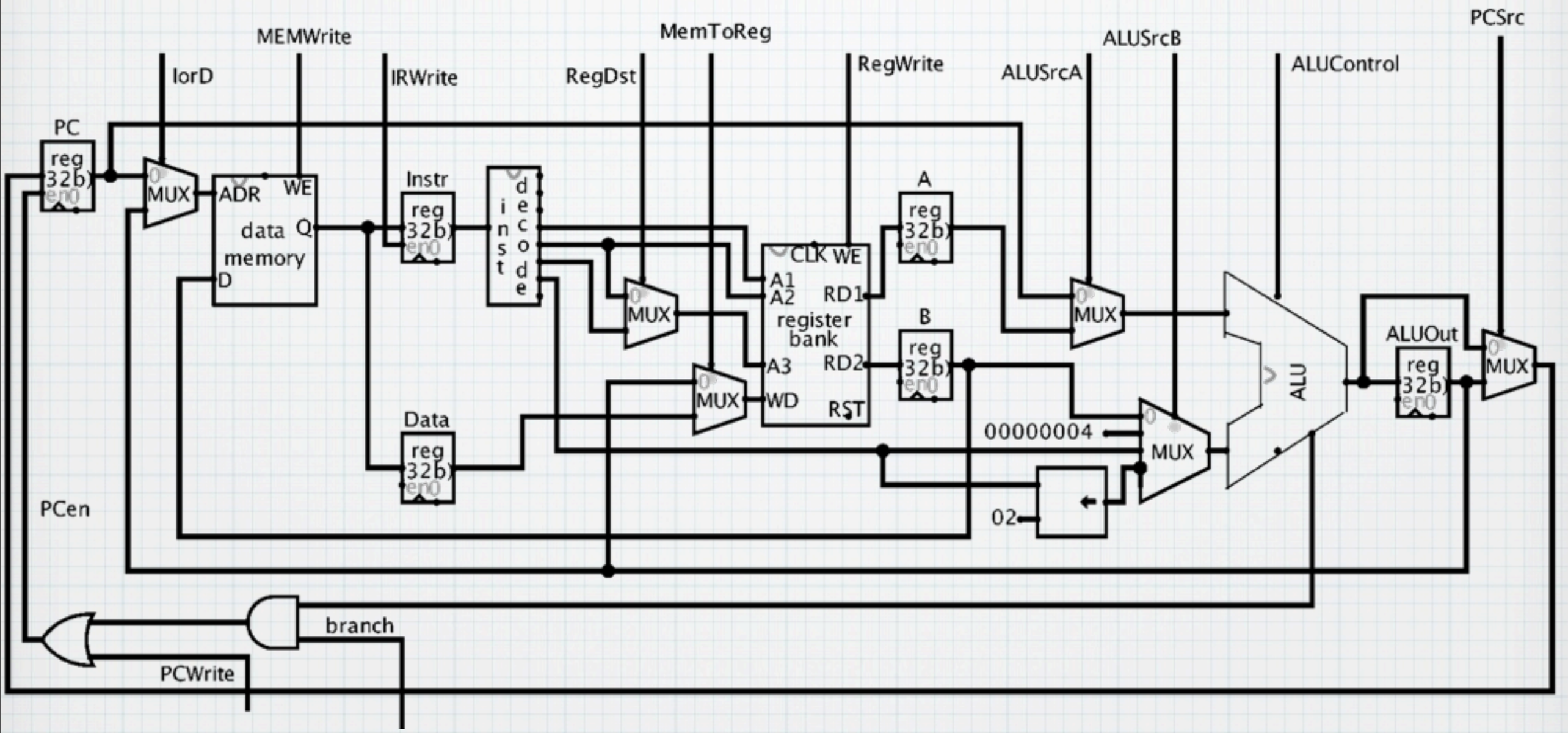
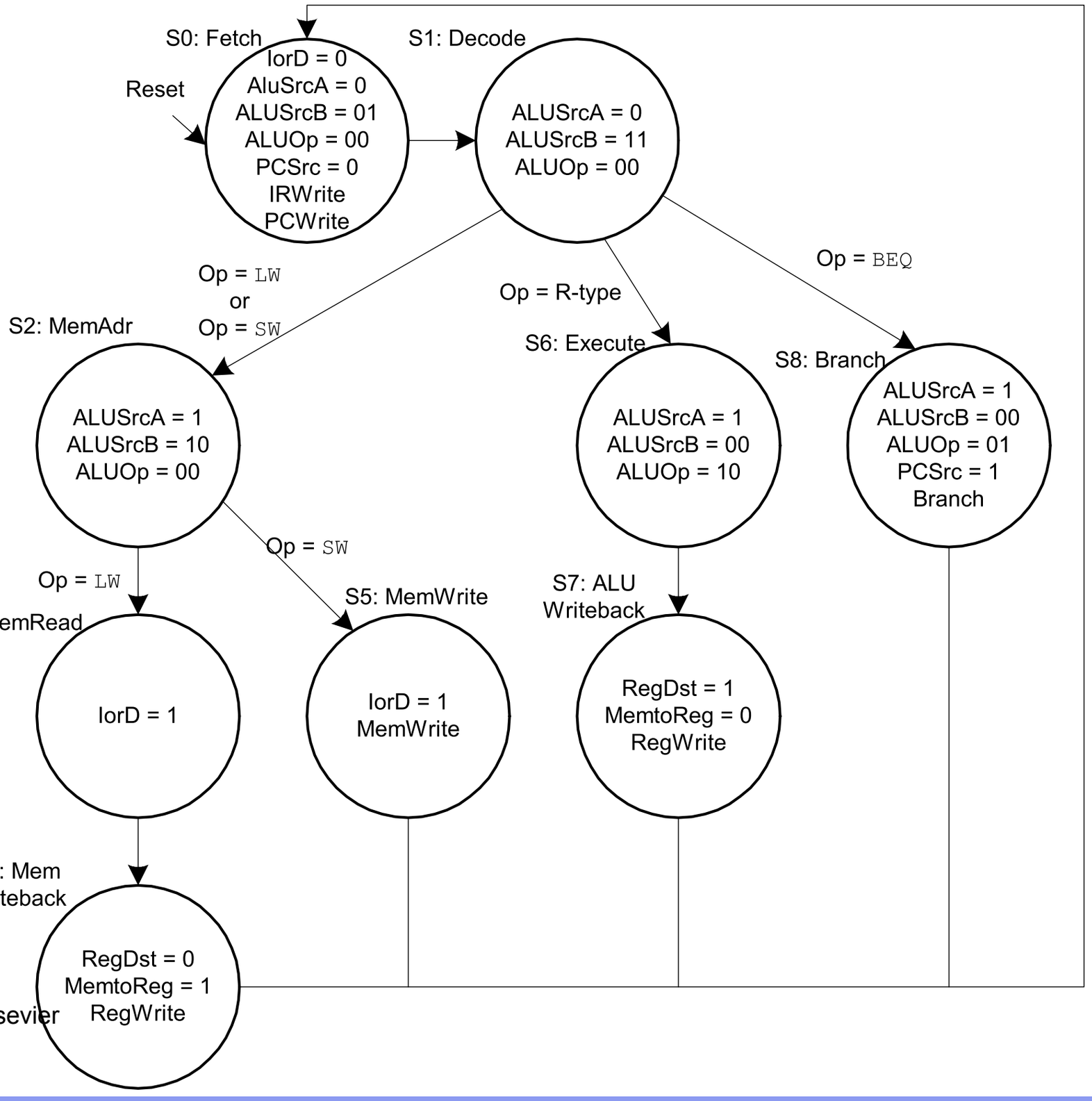
Announcements

TA Application Time

TEPID Handout Updated


IorD
MEMWrite
IRWrite
RegDst
MemToReg
RegWrite
ALUSrcA
ALUSrcB
ALUControl
PCSrc
PC
reg
32b
en
0
MUX
ADR
WE
data
memory
Q
D
Instr
reg
32b
en
0
inst
decode
CLK
WE
A1
A2
RD1
register
bank
A3
RD2
WD
RST
A
reg
32b
en
0
B
reg
32b
en
0
MUX
MUX
MUX
MUX
Data
reg
32b
en
0
00000004
02
ALU
ALUOut
reg
32b
en
0
MUX
PCen
branch
PCWrite

Complete Multicycle Controller FSM



CYCLE 0

IR:= Memory[PC]
PC:= PC+4

CYCLE 1

A:= Reg[IR[25:21]]
B:= Reg[IR[20:16]]
ALUout := PC + sign-extend(IR[15:0]]

CYCLE 2

ALUout:= A + IR[15:0]

if A=B then
PC := ALUout

ALUout:= A op B

CYCLE 3

SW:
Memory[ALUout]:= B

LW:
MDR:= Memory[ALUout]

Reg[IR[15:11]]:= ALUout

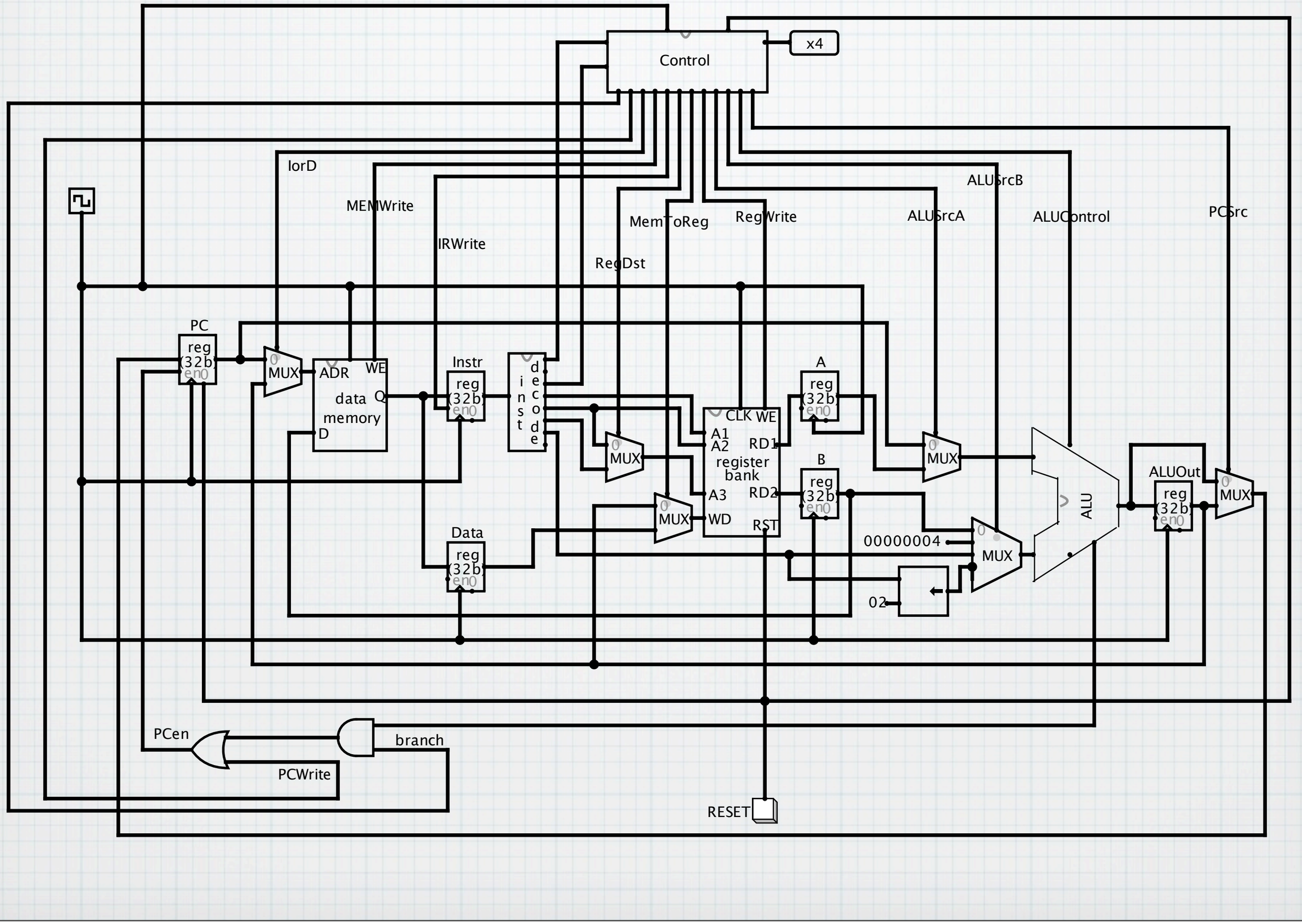
CYCLE 4

Reg[[20:16]]:= MDR

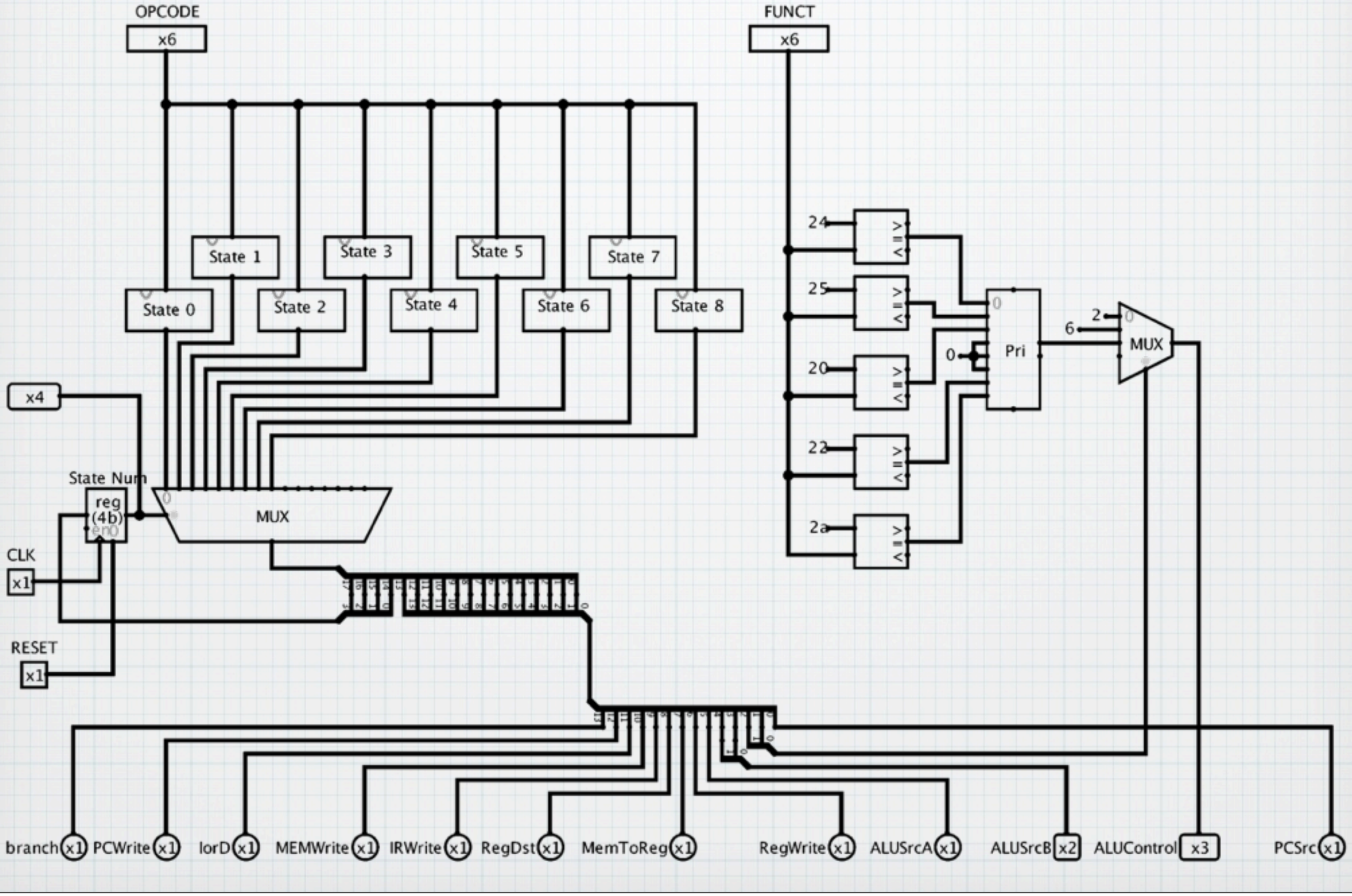
LW AND SW

BRANCH EQ

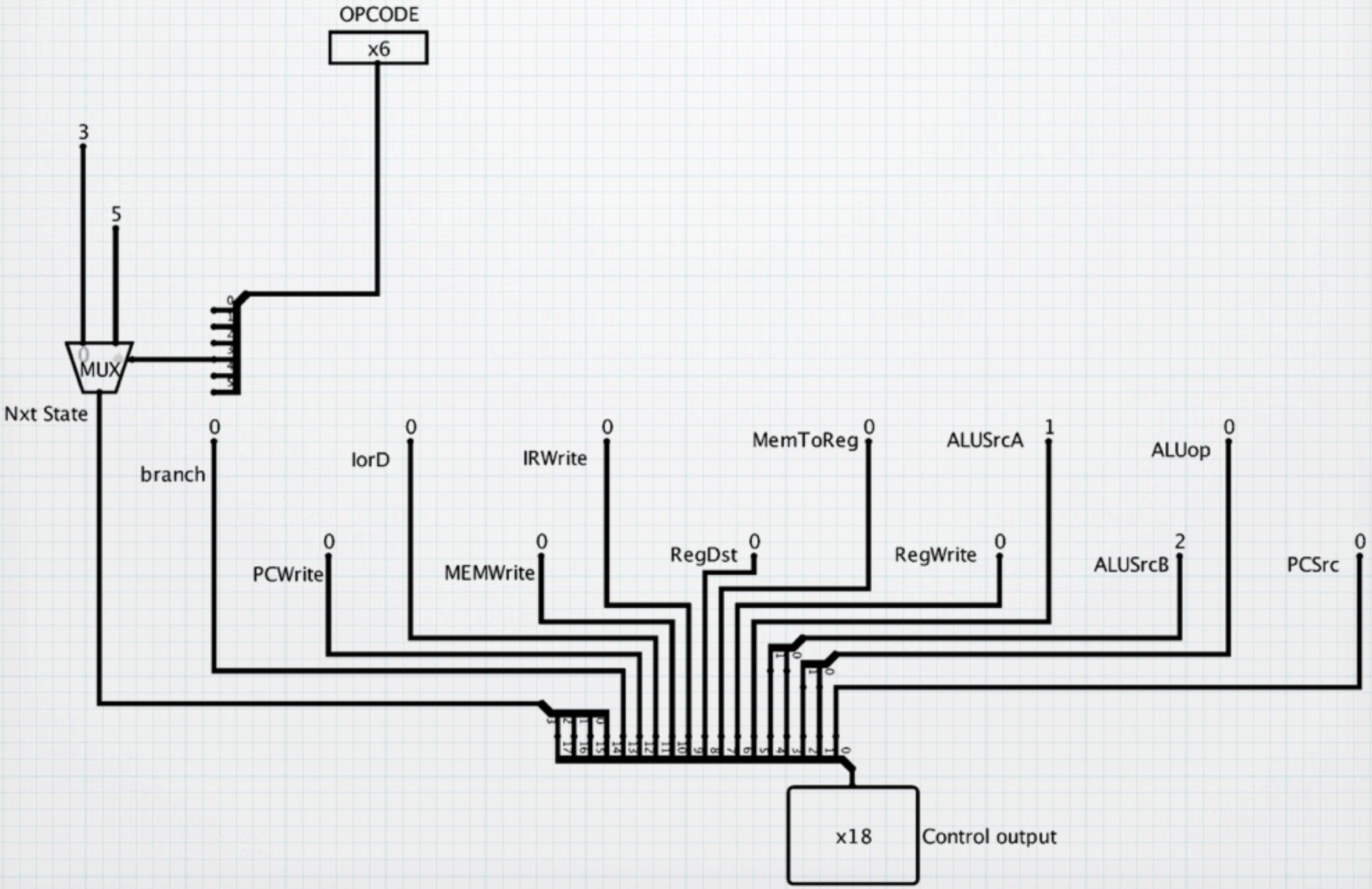
R-TYPE


Control
x4
IorD
MEMWrite
IRWrite
MemToReg
RegDst
RegWrite
ALUSrcA
ALUSrcB
ALUControl
PCSrc
PC
reg
(32b)
en
0
MUX
ADR
WE
data
memory
D
Q
Instr
reg
(32b)
en
inst
decode
CLK
WE
A1
A2
RD1
register
bank
A3
RD2
WD
RST
A
B
ALU
ALUOut
00000004
02
Data
PCen
branch
PCWrite
RESET

Implementing Multi-state Contro



Implementing Multi-state Contro



(Partial) Loop Unrolling

```
for ( I = exp1; I < exp2; i++ )
        A(I)
```

⟹

```
for ( I = exp1; I < exp2 ; I=I+2 )
        A(I)
        A(I+1)
```

(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i++ ) {
      a[i] = 0;
}
```

⟹

```
for ( i = 0; i < 1000; i = i + 2 ) {
      a[i] = 0;
      a[i+1] = 0;
}
```

(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i++ ) {
      a[i] = 0;
}
```

```
    la  $t0,a
#    for(int i=0;i<1000;i++)
    li   $t3,0
    li   $t4,1000
zeroloop:
    slt $t5,$t3,$t4
    beq   $t5,$0,zeroloopEnd

#    a[i] = 0;
    add   $t6,$t3,$t0
    sw $0,0($t6)


    addi  $t3,$t3,1
    b  zeroloop:
zeroloopEnd:
```

(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i = i + 2 ) {
    a[i] = 0;
    a[i+1] = 0;
}
```

```
    la  $t0,a
#   for(int i=0;i<1000;i=i+2)
    li  $t3,0
    li  $t4,1000
zeroloop2:
    slt $t5,$t3,$t4
    beq  $t5,$0,zeroloop2End

#   a[i] = 0;
    add  $t6,$t3,$t0
    sw $0,0($t6)
    sw $0,4($t6)

    addi  $t3,$t3,2
    b  zeroloop2:
zeroloopEnd2:
```

(Partial) Loop Unrolling

```
for ( i = 0; i < 1000; i = i + 2 ) {
    a[i] = 0;
    a[i+1] = 0;
}
```

6*1000 = 6000

vs.

7*500 = 3500

vs

9*250 = 2250

```
    la  $t0,a
#   for(int i=0;i<1000;i=i+2)
    li  $t3,0
    li  $t4,1000
zeroloop2:
    slt $t5,$t3,$t4
    beq  $t5,$0,zeroloop2End

#   a[i] = 0;
    add  $t6,$t3,$t0
    sw $0,0($t6)
    sw $0,4($t6)

    addi  $t3,$t3,2
    b  zeroloop2:
zeroloopEnd2:
```

Loop Merge

```
for ( Ia = exp1; Ia < exp2; Ia++ )
                    A(Ia)
for ( Ib = exp1; Ib < exp2; Ib++ )
                    B(Ib)
```

⇒

```
for ( I = exp1; I <exp2; I++)
                    A(I)
                    B(I)
```

Loop Merge

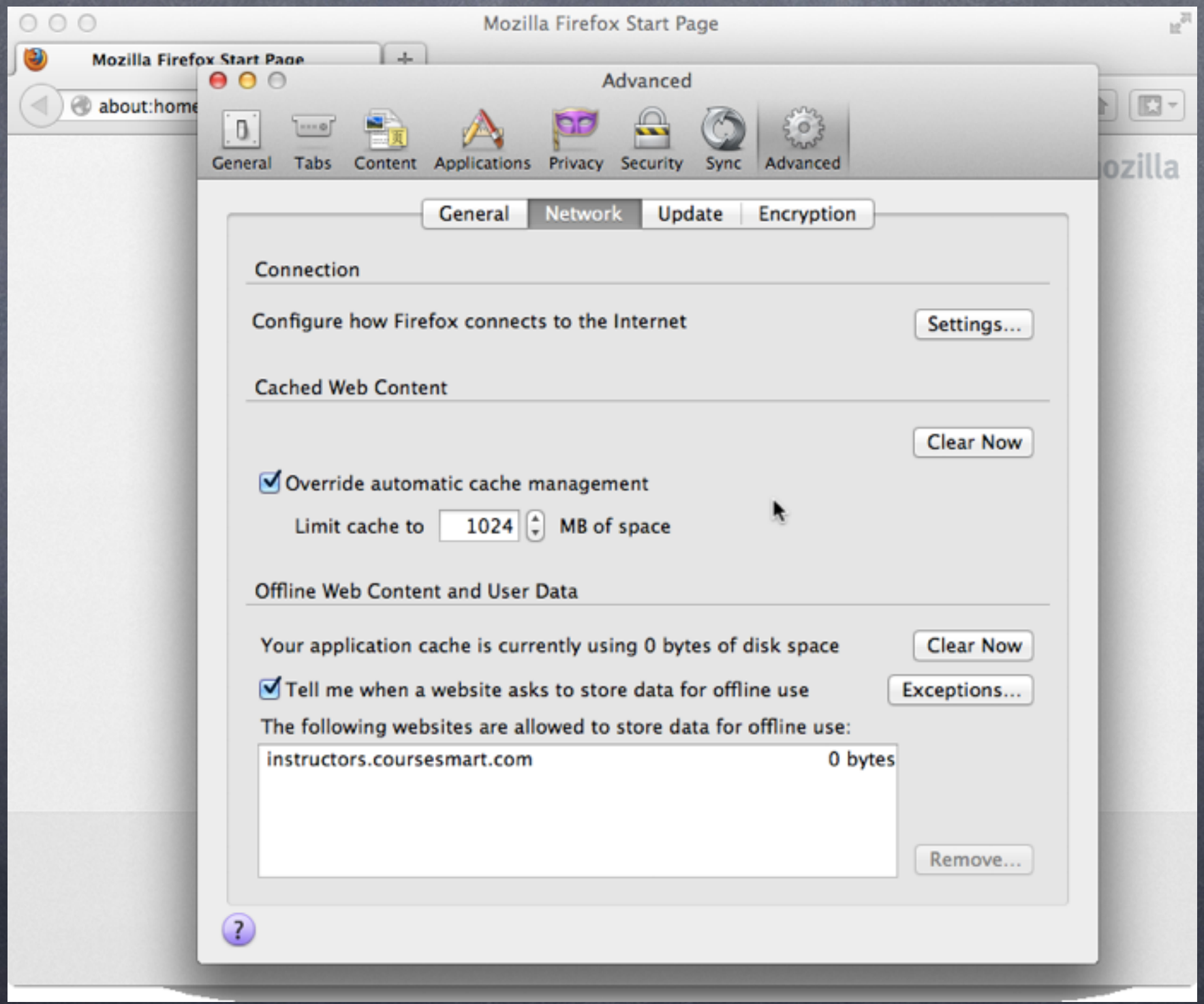
```
for ( i = 0; i < 1000; i++ ) {
      a[i] = 0;
}
for ( i = 0; i < 1000; i++ ) {
      b[i] = 0;
}
```

⟹

```
for ( i = 0; i < 1000; i++ ) {
      a[i] = 0;
      b[i] = 0;
}
```

Cash, Cache, or Cachet?

Cash, Cache, or Cachet?



Cash, Cache, or Cachet?

Processing

12-core (standard configuration)

- Two 2.4GHz 6-Core Intel Xeon E5645 processors
- 12MB of fully shared L3 cache per processor
- Turbo Boost dynamic performance up to 2.67GHz
- Hyper-Threading technology for up to 24 virtual cores

Quad-core (standard configuration)

- One 3.2GHz Quad-Core Intel Xeon W3565 processor
- 8MB of fully shared L3 cache per processor
- Turbo Boost dynamic performance up to 3.46GHz
- Hyper-Threading technology for up to 8 virtual cores

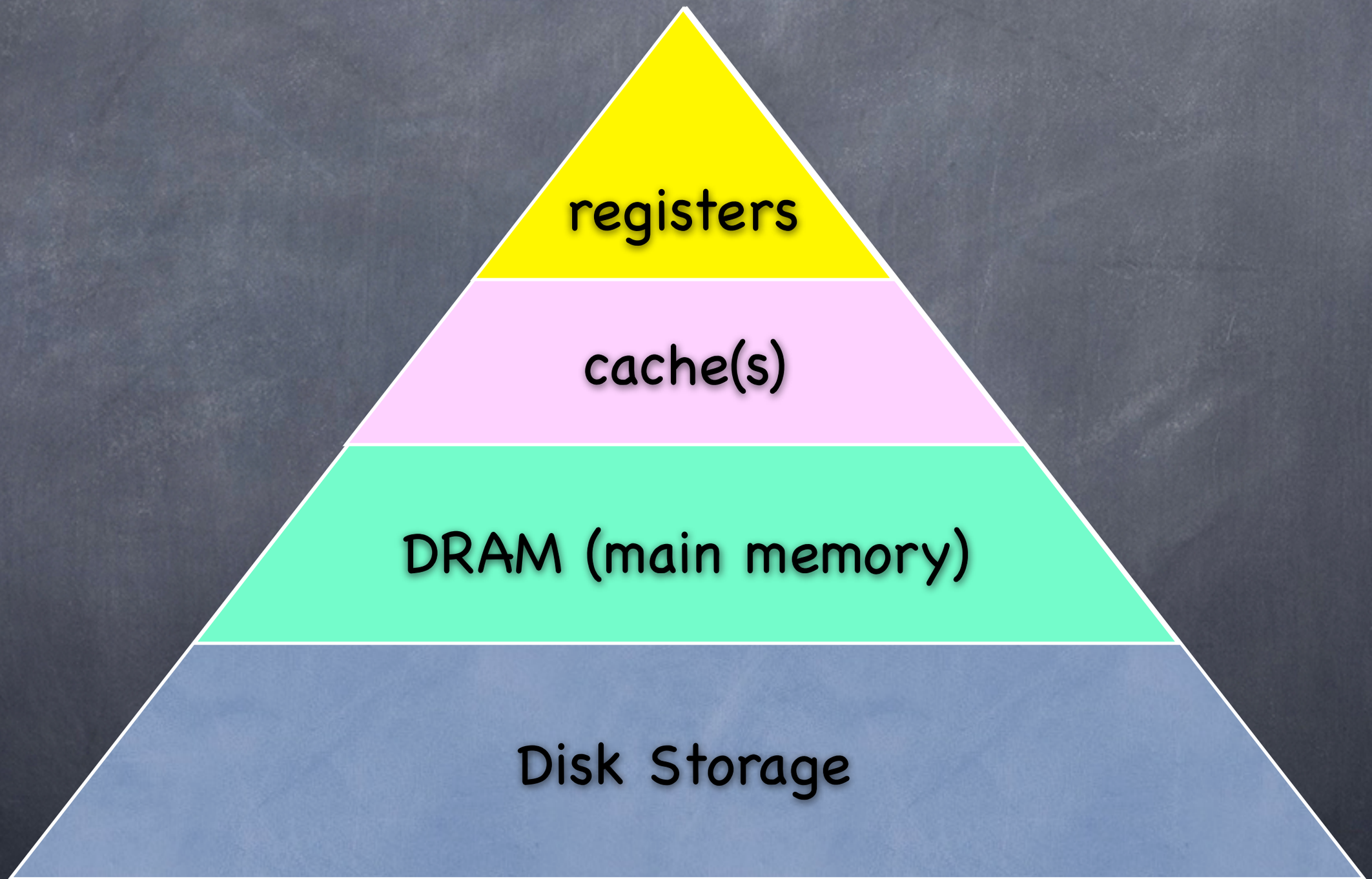
Principle(s) of Locality

- Temporal Locality:
  - Locality in time (e.g., if looked at a Web page recently, likely to look at it again soon)
  - If data used recently, likely to use it again soon
  - Keep recently accessed data in more easily accessed storage

Principle(s) of Locality

- Spatial Locality:
  - Locality in space (e.g., if read one page of book, likely to read nearby pages soon)
  - If data used recently, likely to use nearby data soon
  - When accessing data, bring nearby data into easily accessed storage too.

Memory Hierarchy



Memory Hierarchy

10^0 cycles

registers

< 10^1 cycles

cache(s)

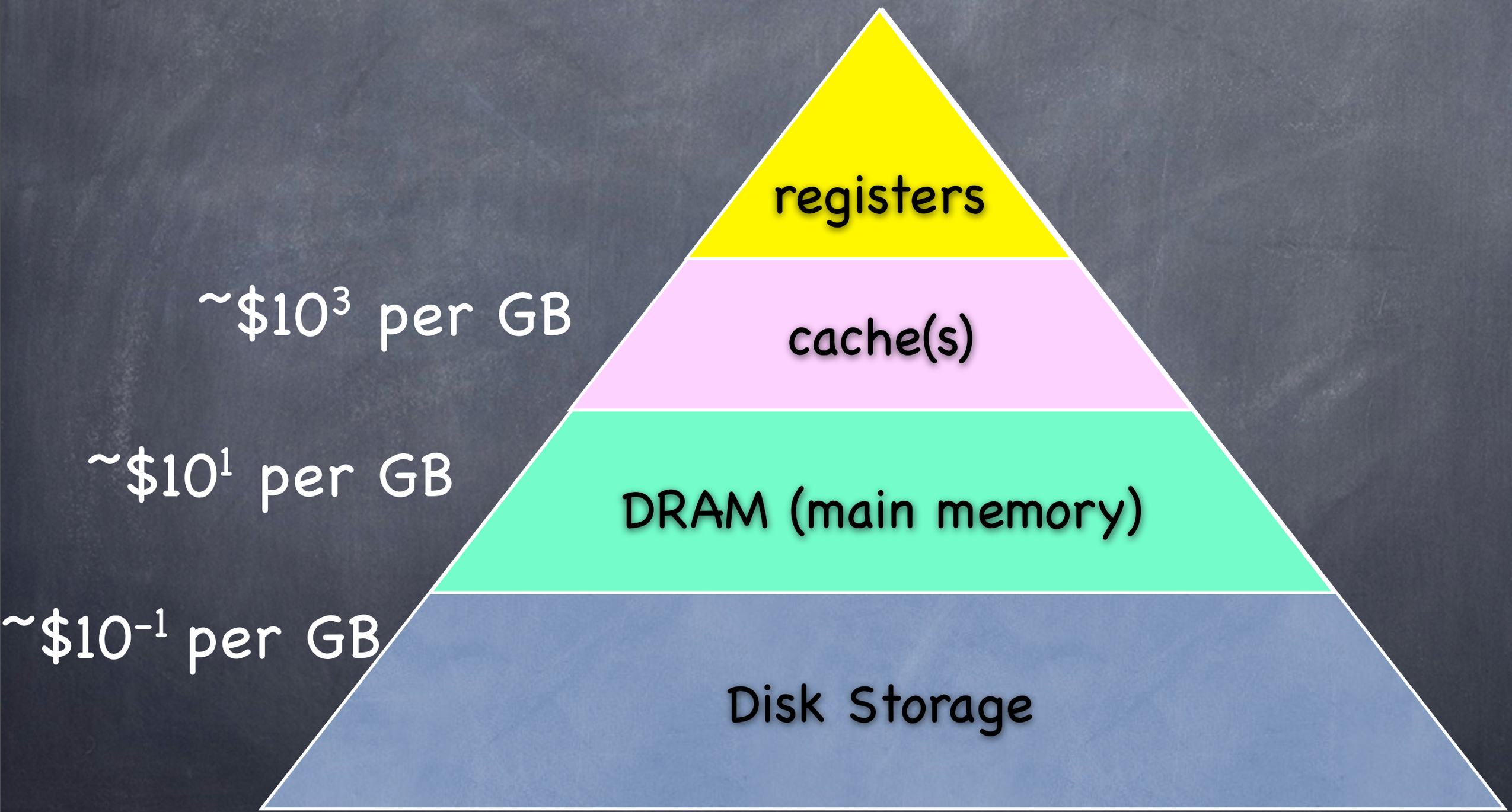
< 10^2 cycles

DRAM (main memory)

10^6 words

Disk Storage

Memory Hierarchy



Memory Hierarchy

10^2 words

registers

10^6 words

cache(s)

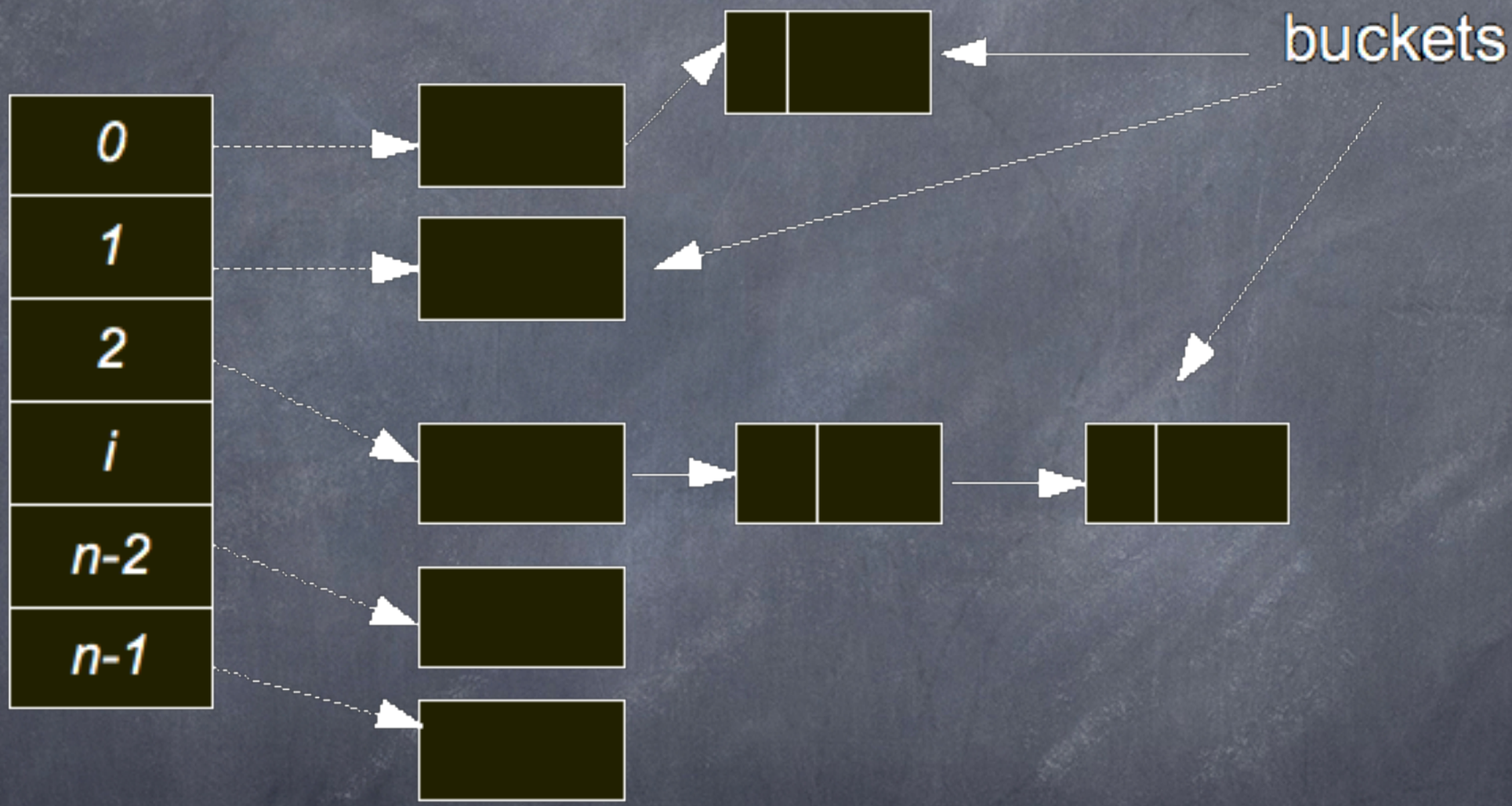
10^9 words

DRAM (main memory)

10^{12} words

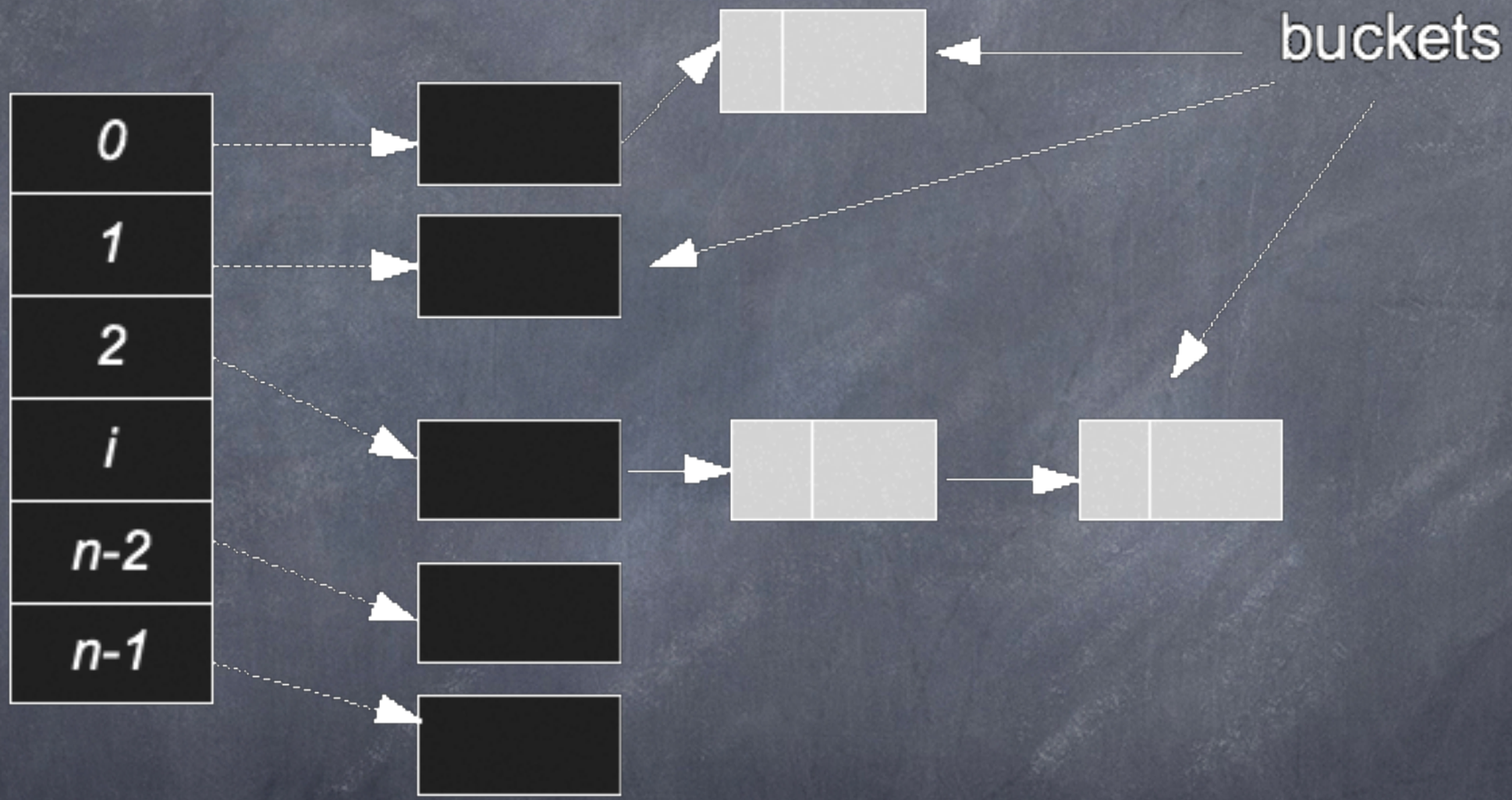
Disk Storage

Hash vs. Cache



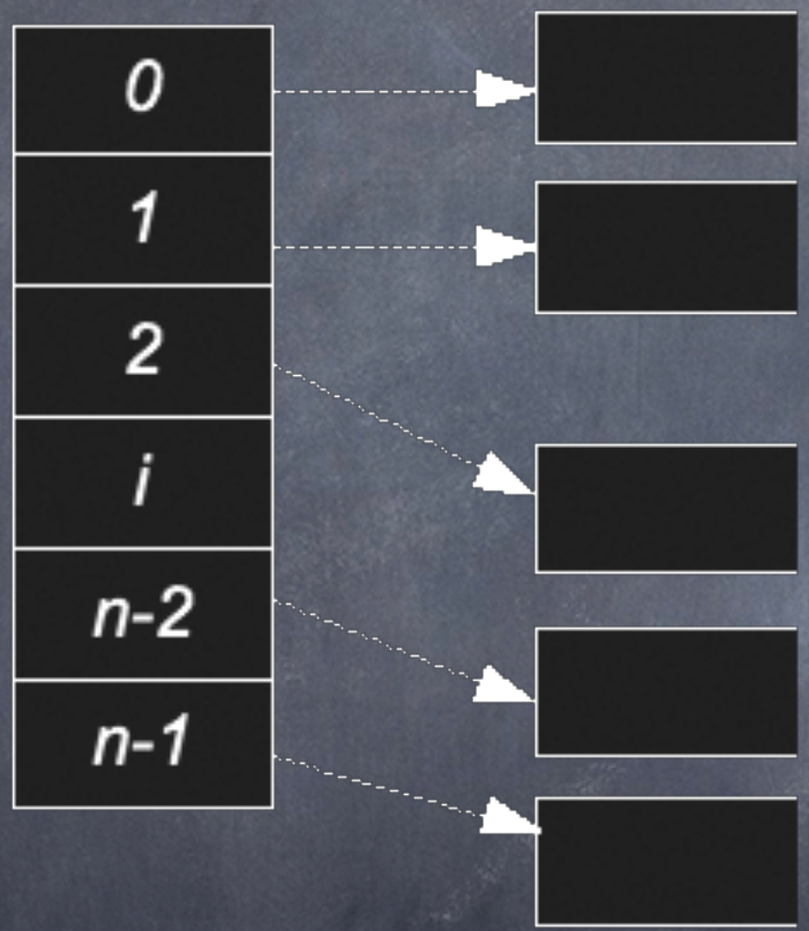
- Hash function maps keys to ``random'' bins
- Lookup within bins by linked list traversal

Forgetful Hash vs. Cache



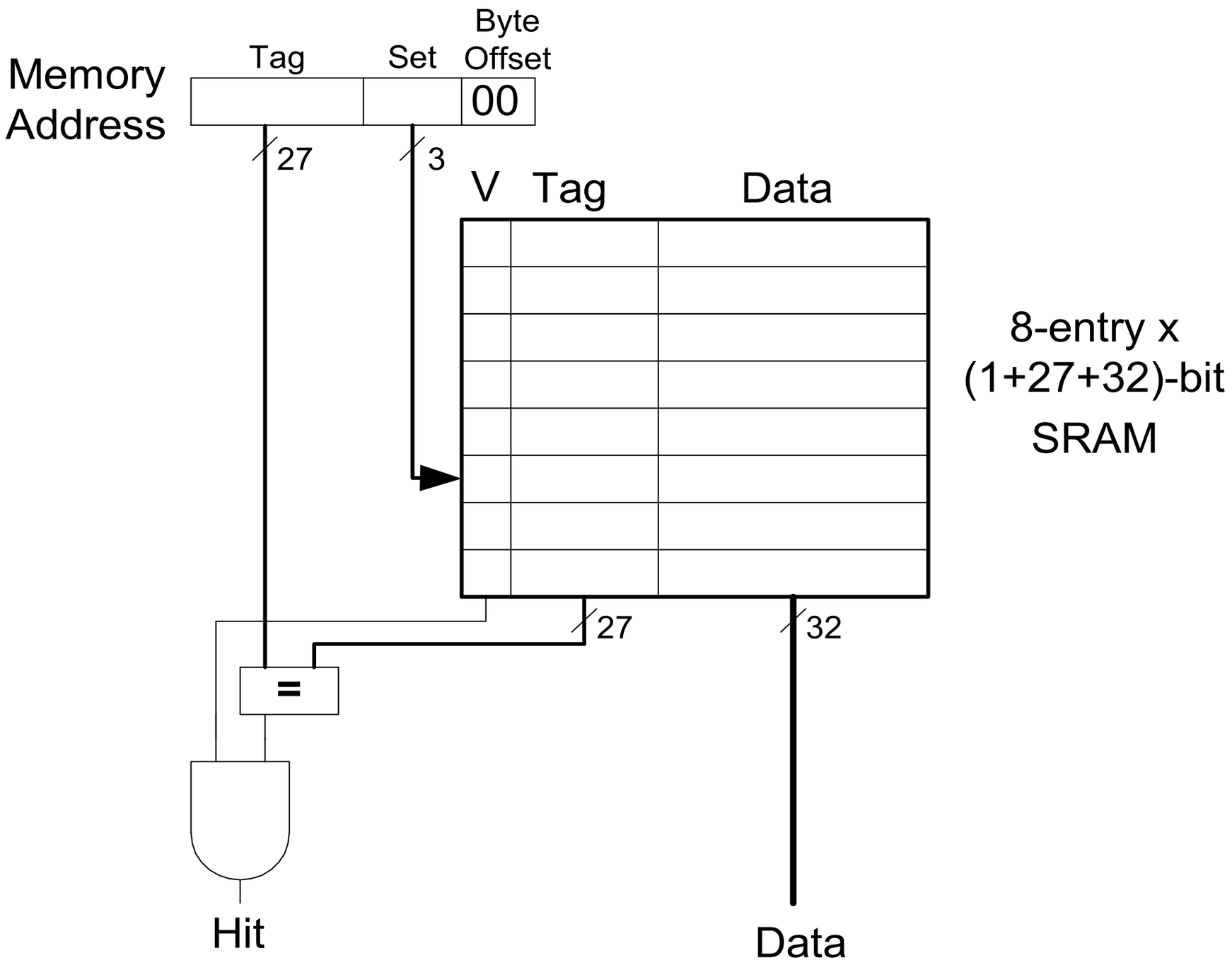
- Only keep first item in each hash bin list

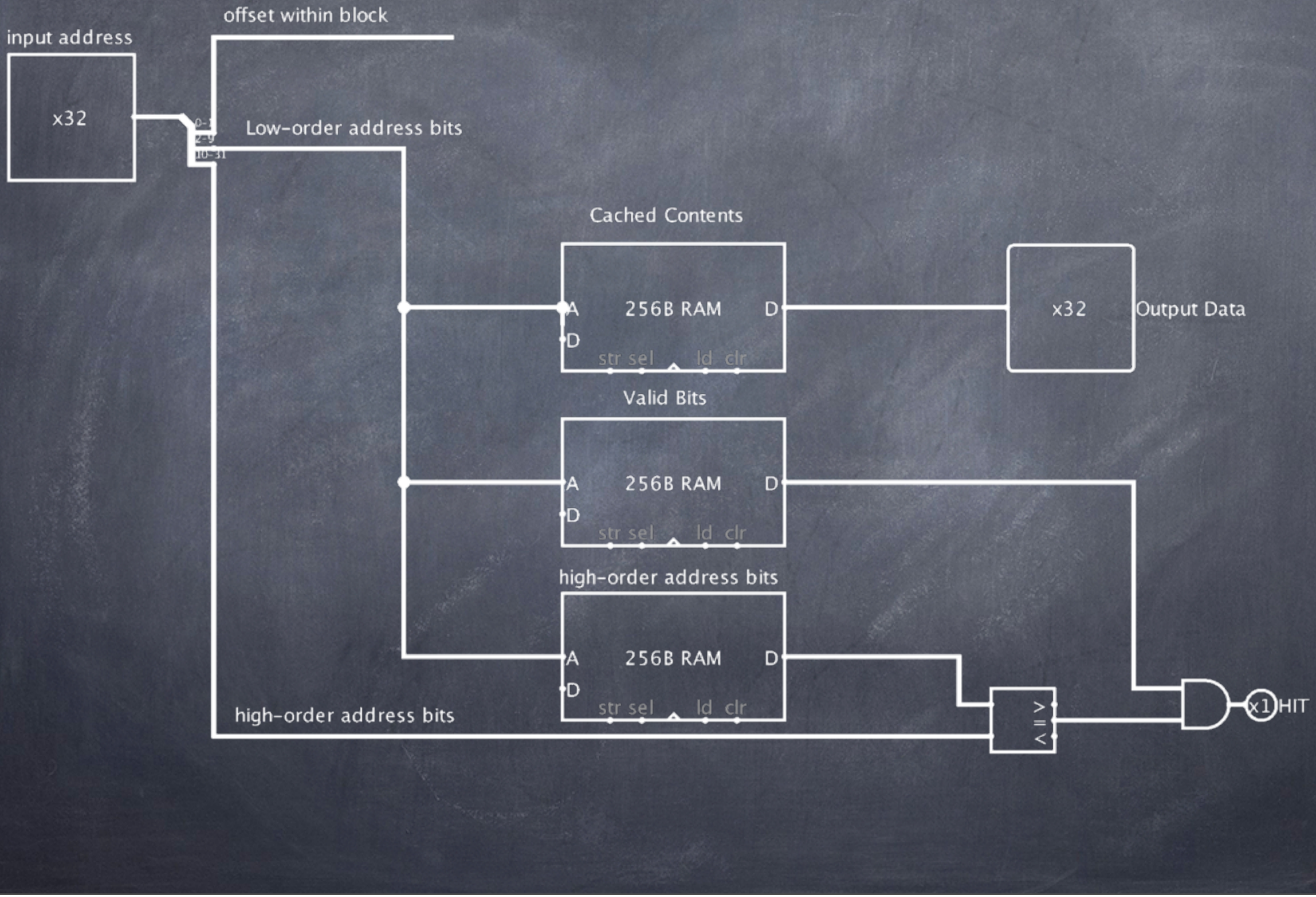
Cache

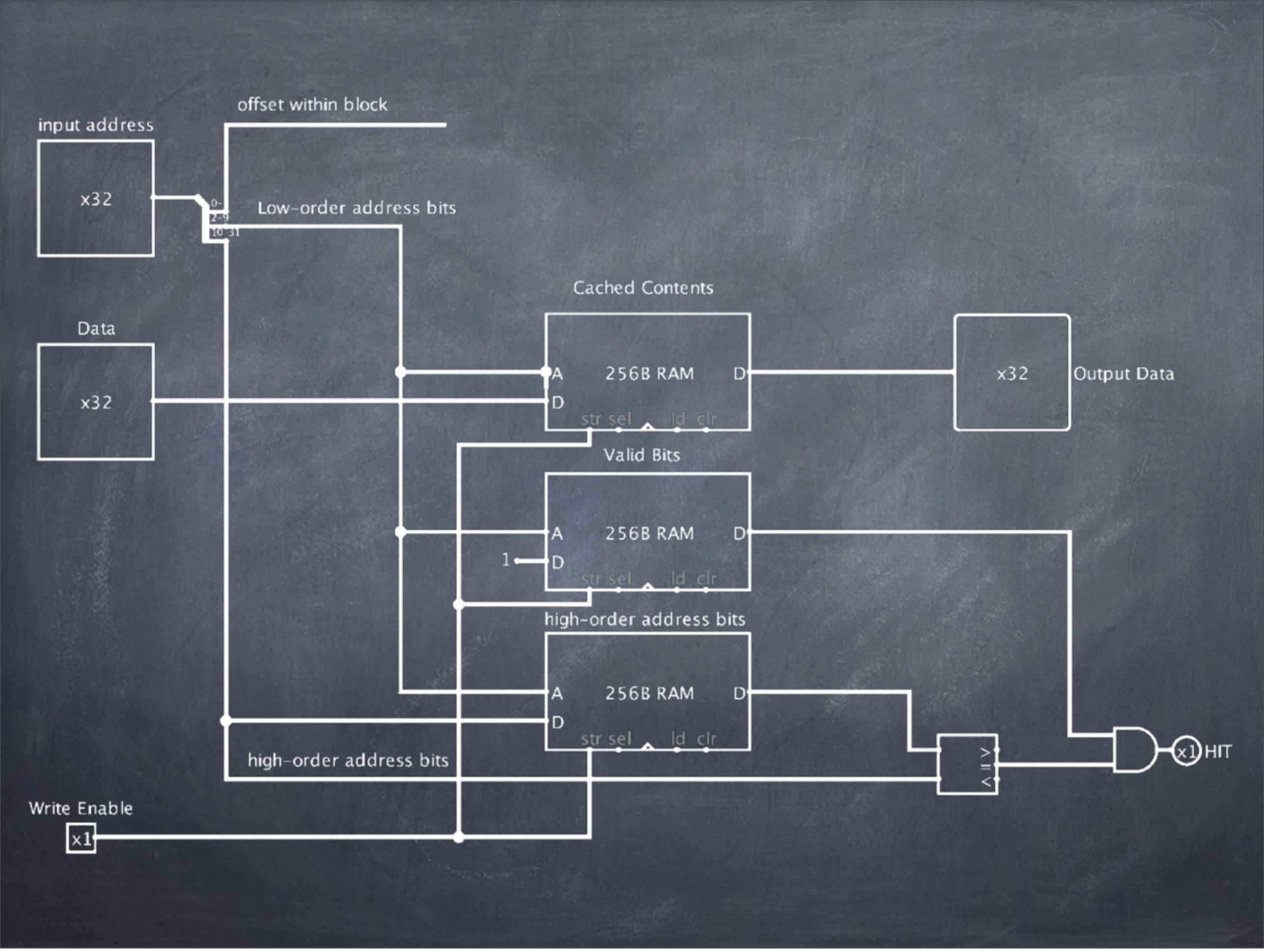


- Hash function is suffix of address / cache unit size
- Fixed size bins allows fast search.

Direct Mapped Cache Hardware




input address
x32
offset within block
0-1
2-9
10-31
Low-order address bits
Cached Contents
A
D
256B RAM
D
str sel
ld clr
x32
Output Data
Valid Bits
A
D
256B RAM
D
str sel
ld clr
high-order address bits
A
D
256B RAM
D
str sel
ld clr
high-order address bits
>
=
<
x1
HIT


offset within block
input address
x32
0-
2-9
10 31
Low-order address bits
Cached Contents
Data
x32
A
256B RAM
D
D
str sel
ld clr
x32
Output Data
Valid Bits
A
256B RAM
D
1
D
str sel
ld clr
high-order address bits
A
256B RAM
D
D
str sel
ld clr
high-order address bits
A = V
x1
HIT
Write Enable
x1

Cache Terminology

Capacity (C):

the number of data bytes a cache stores

Block size (b):

bytes of data brought into cache at once

Number of blocks ($B = C/b$):

number of blocks in cache: $B = C/b$

Degree of associativity (N):

number of blocks in a set

Number of sets ($S = B/N$):

each memory address maps to exactly one cache set