CS 3410

Computer System Organization and Programming

K. Walsh
kwalsh@cs

TAs:
Deniz Altinbuken
Hussam Abu-Libdeh

Consultants:
Adam Sorrin
Arseney Romanenko

If you want to make an apple pie from scratch, you must first create the universe.

– Carl Sagan

C

```
int x = 10;
x = 2 * x + 15;
```

compiler

MIPS assembly language

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

assembler

MIPS machine language

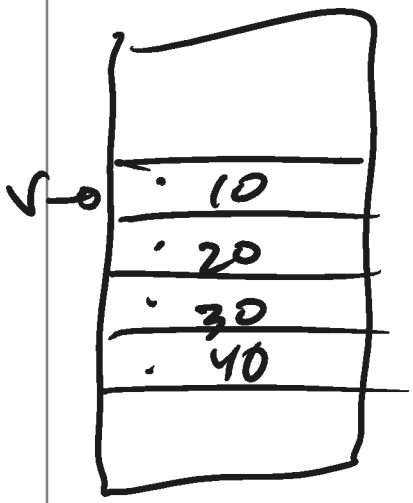
```
00100000000000101000000000001010
00000000000001010010100001000000
00100000101001010000000000001111
```

C

compiler

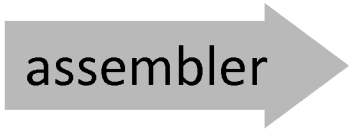
MIPS assembly language

```
int sum3(int v[]) {
  return v[0] +
         v[1] +
         v[2];
}

main() {
  ...
  int v[] = ...;
  int a = sum3(v);
  v[3] = a;
  ...
}
```




```
sum3:
     lw   r9, 0(r5)
     lw   r10, 4(r5)
     lw   r11, 8(r5)
     add  r3, r9, r10
     add  r3, r3, r11
     jr   r31

main:
     ...
     addi r5, r0, 1000
     jal  sum3
     sw   r3, 12(r5)
     ...
```

MIPS assembly language



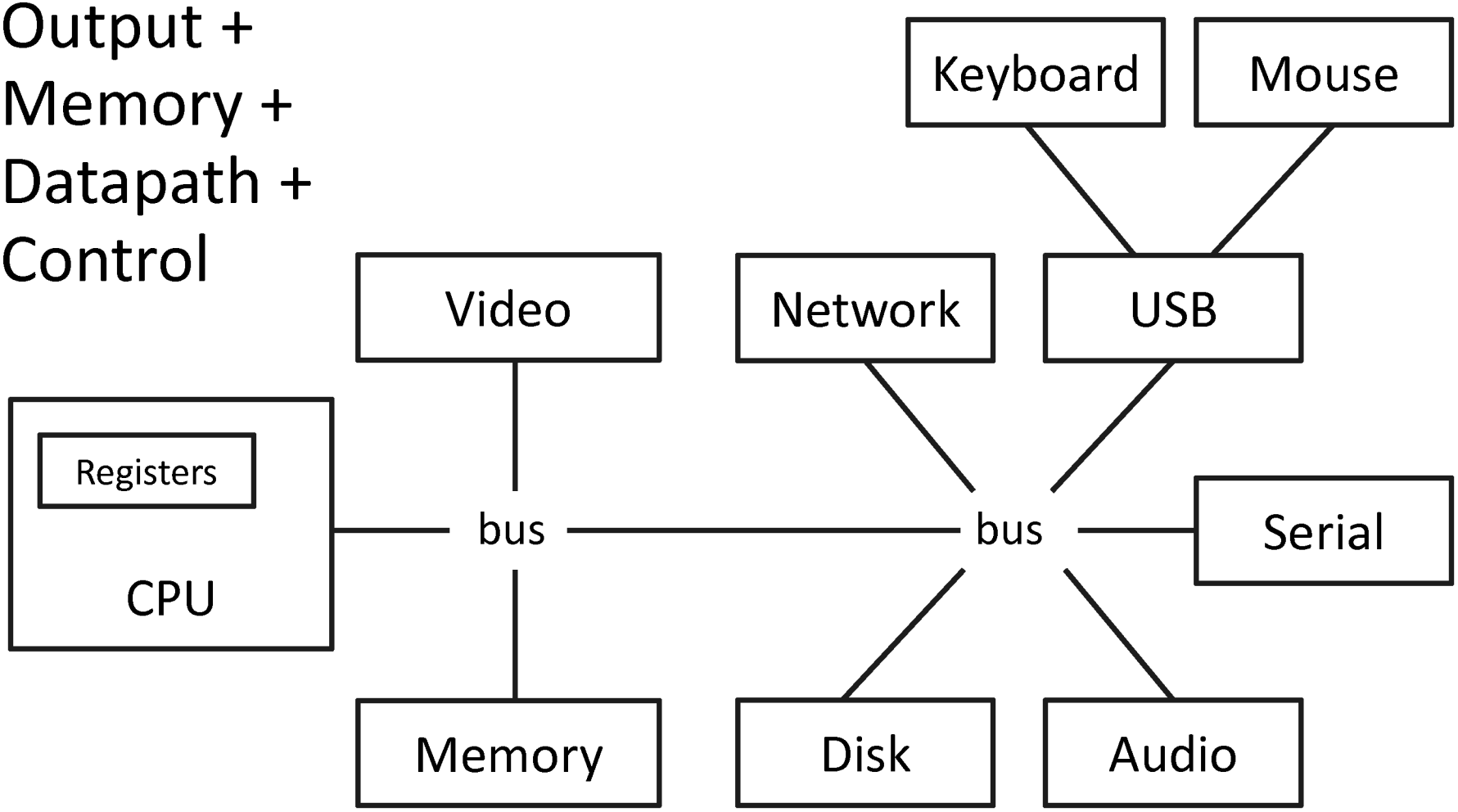
MIPS machine language

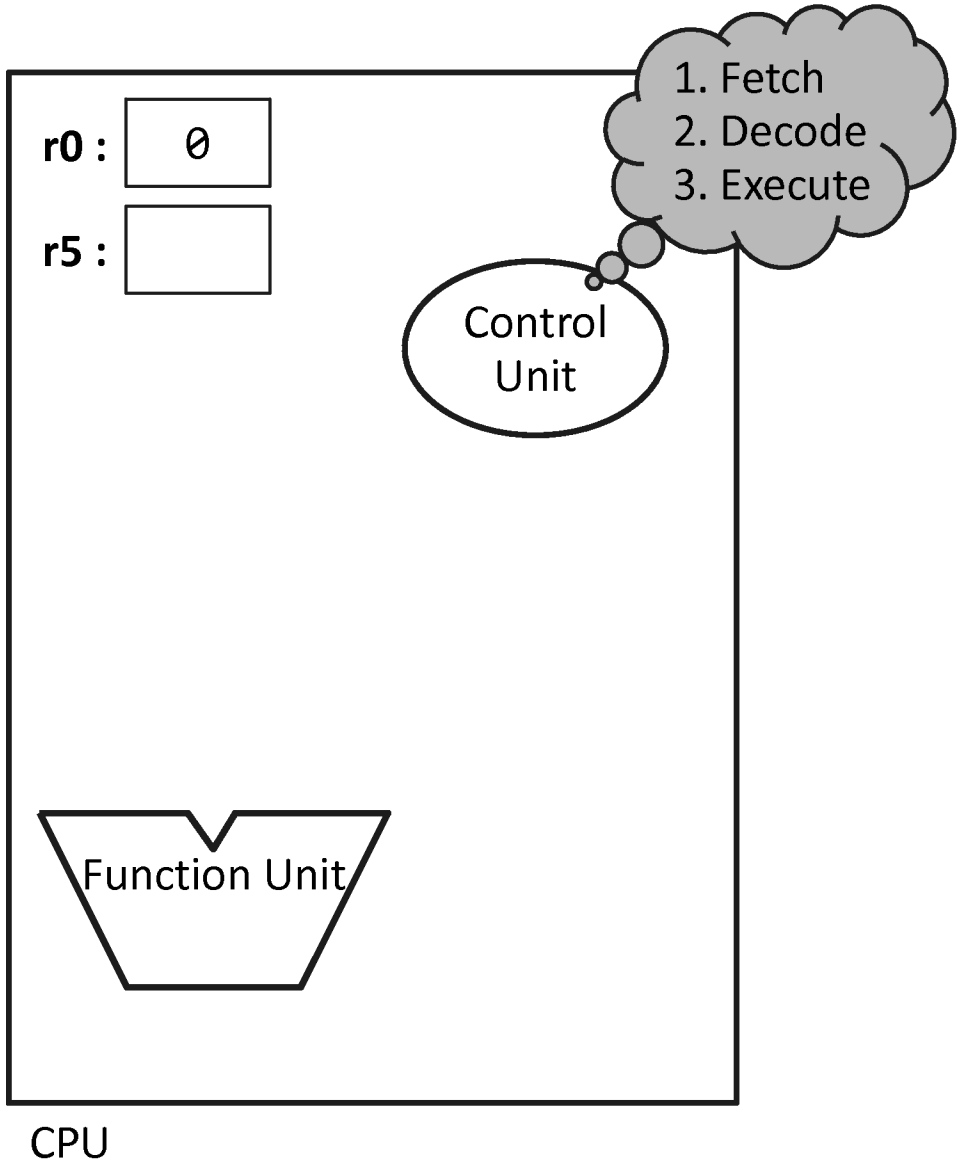
```
sum3:
   lw   r9, 0(r5)
   lw   r10, 4(r5)
   lw   r11, 8(r5)
   add  r3, r9, r10
   add  r3, r3, r11
   jr   r31

main:
   ...
   addi r5, r0, 1000
   jal  sum3
   sw   r3, 12(r5)
   ...
```

```
10001100101010010000000000000000
10001100101010100000000000000100
10001100101010110000000000001000
00000001001010100001100000100000
00000000011010110001100000100000
00000011111000000000000000001000
...
...
...
00100000000001010000001111101000
00001100000100000000000000000000
10101100101000110000000000001100
...
```

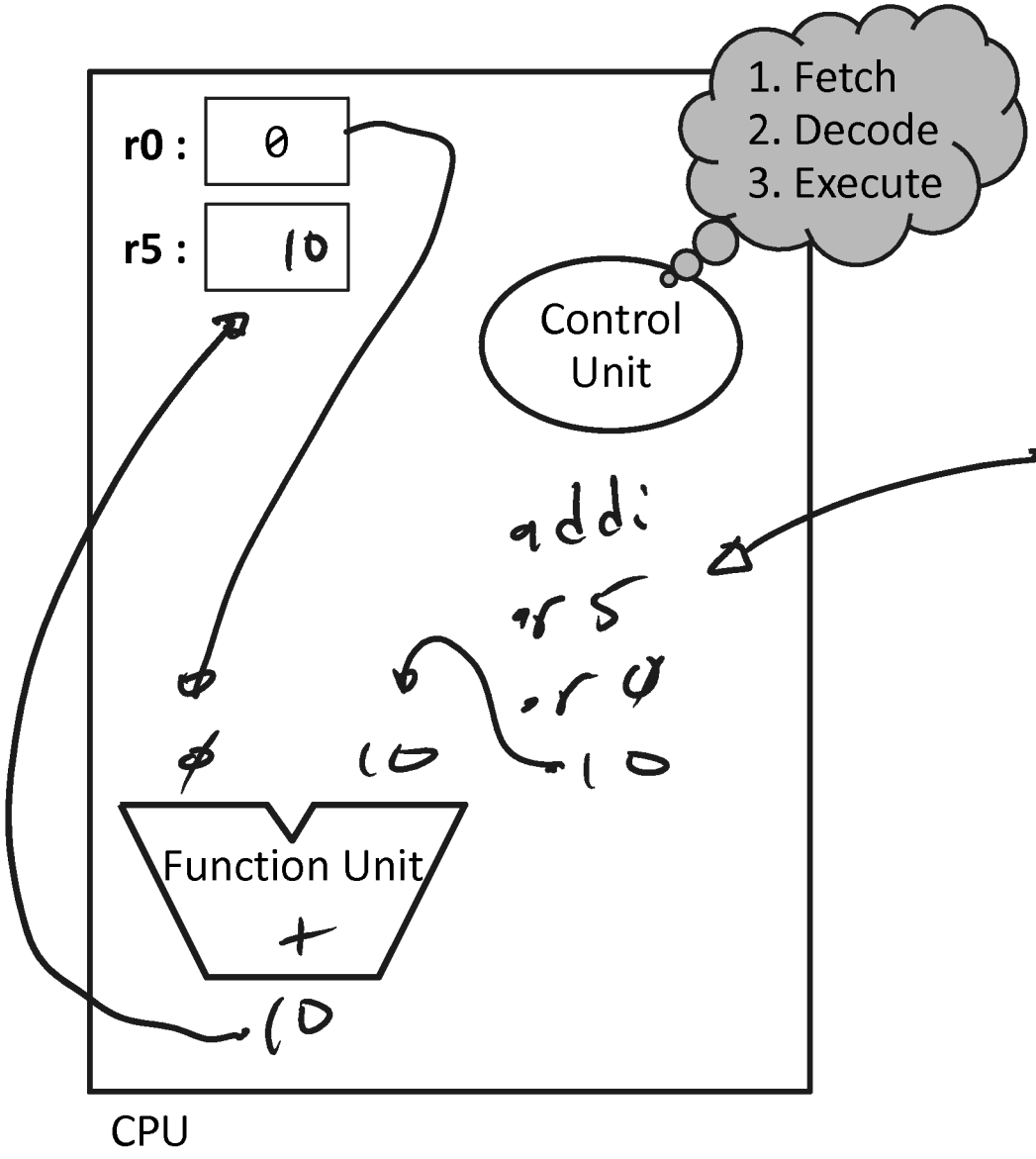
Computer System = ?

Input +
Output +
Memory +
Datapath +
Control

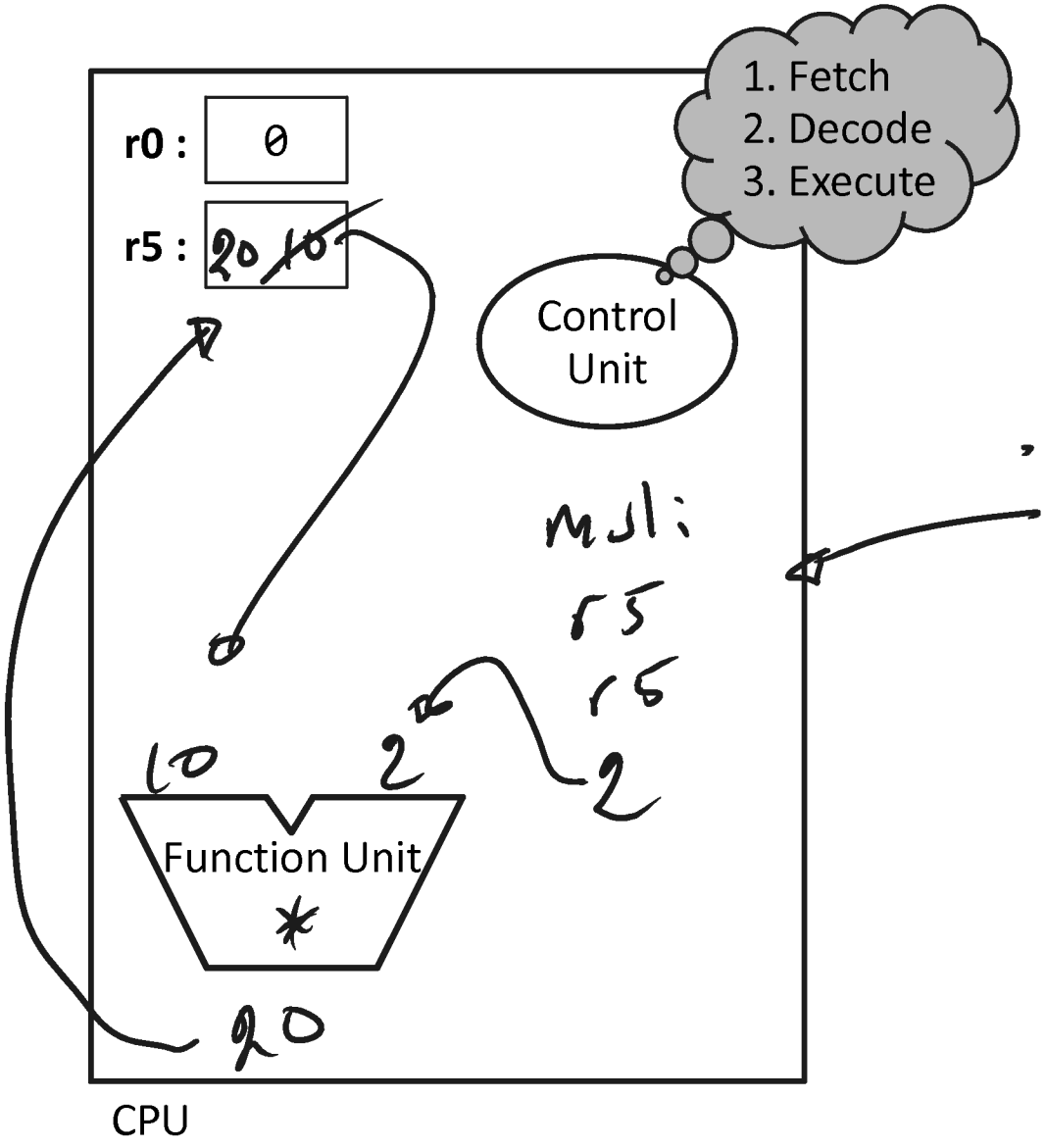




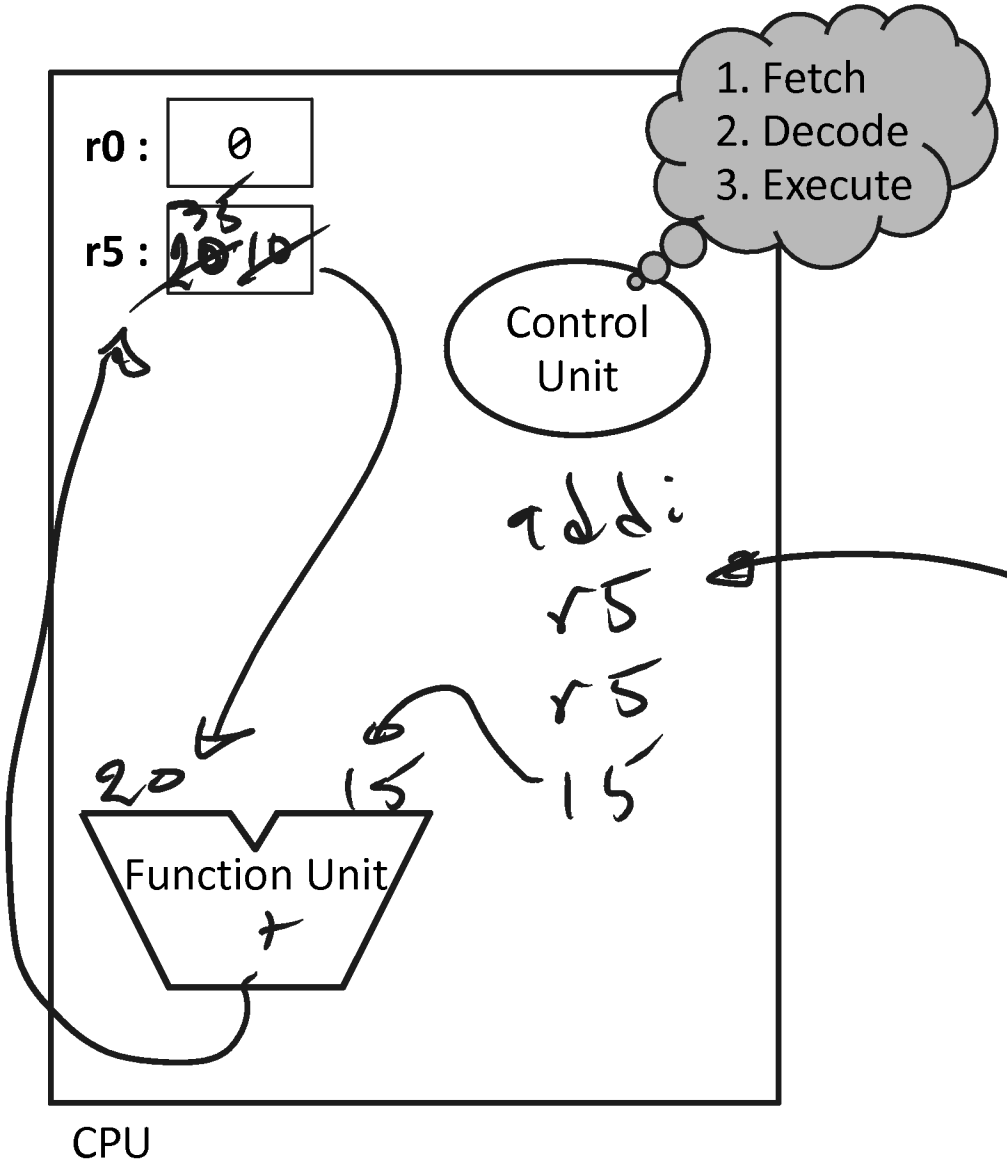

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```



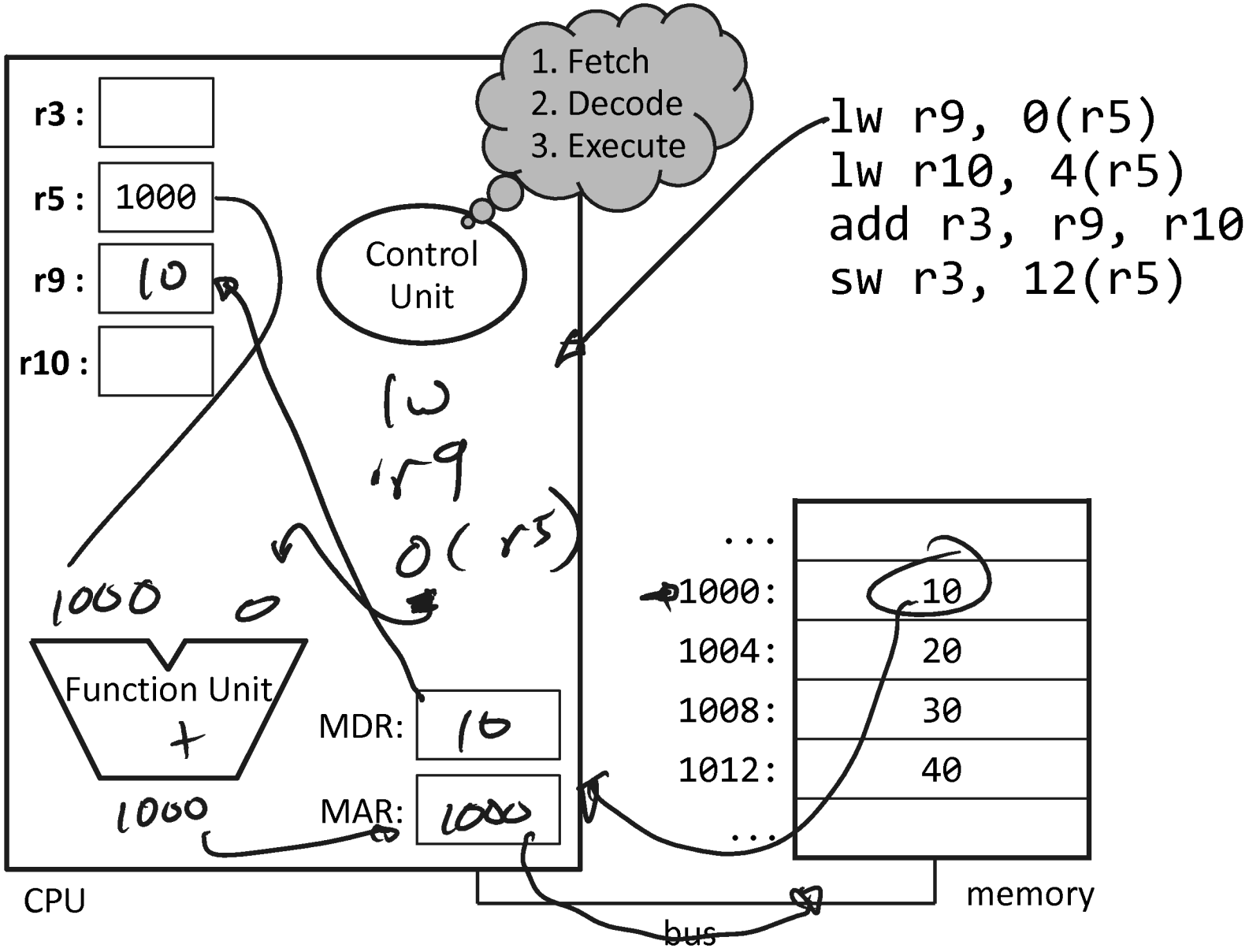

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

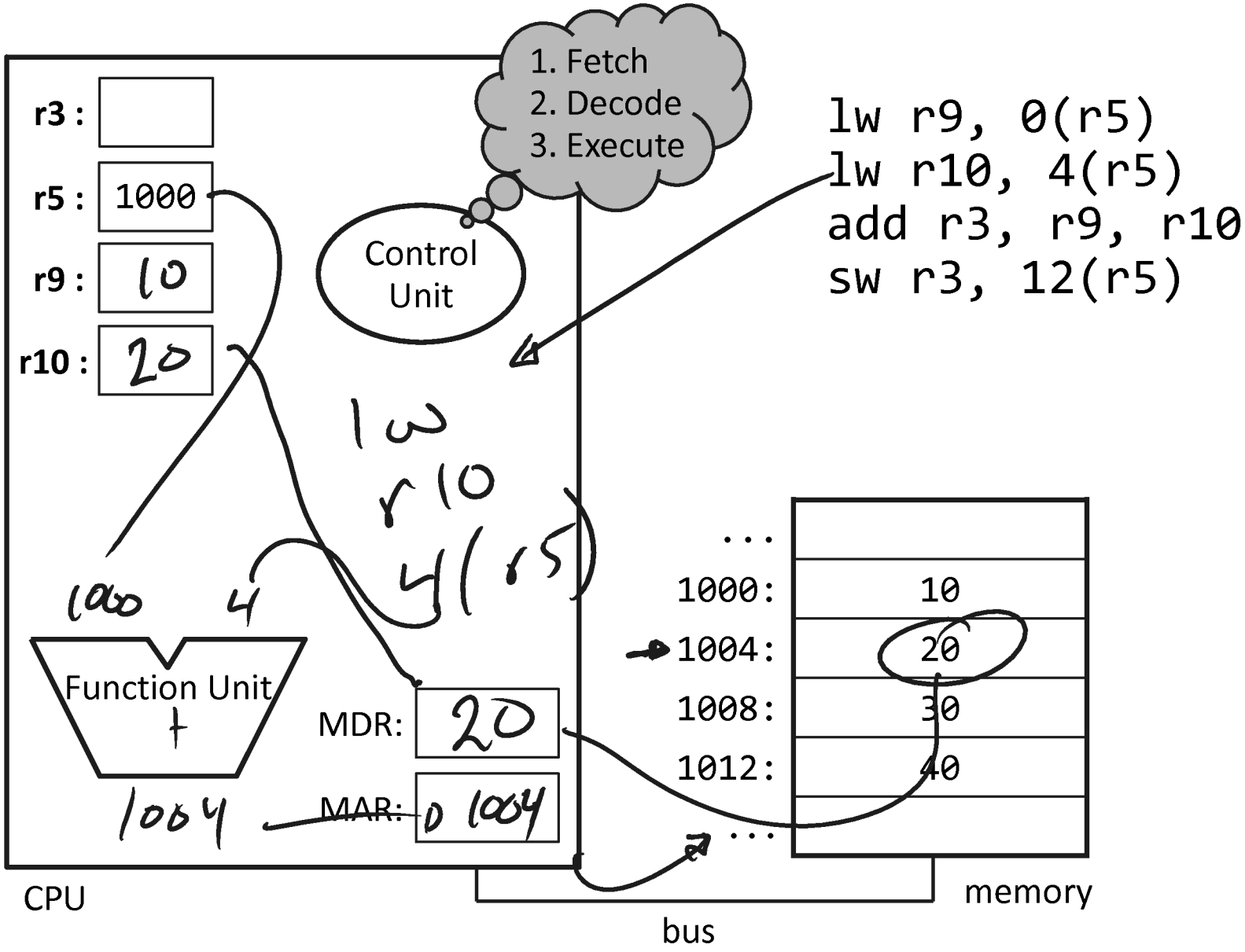


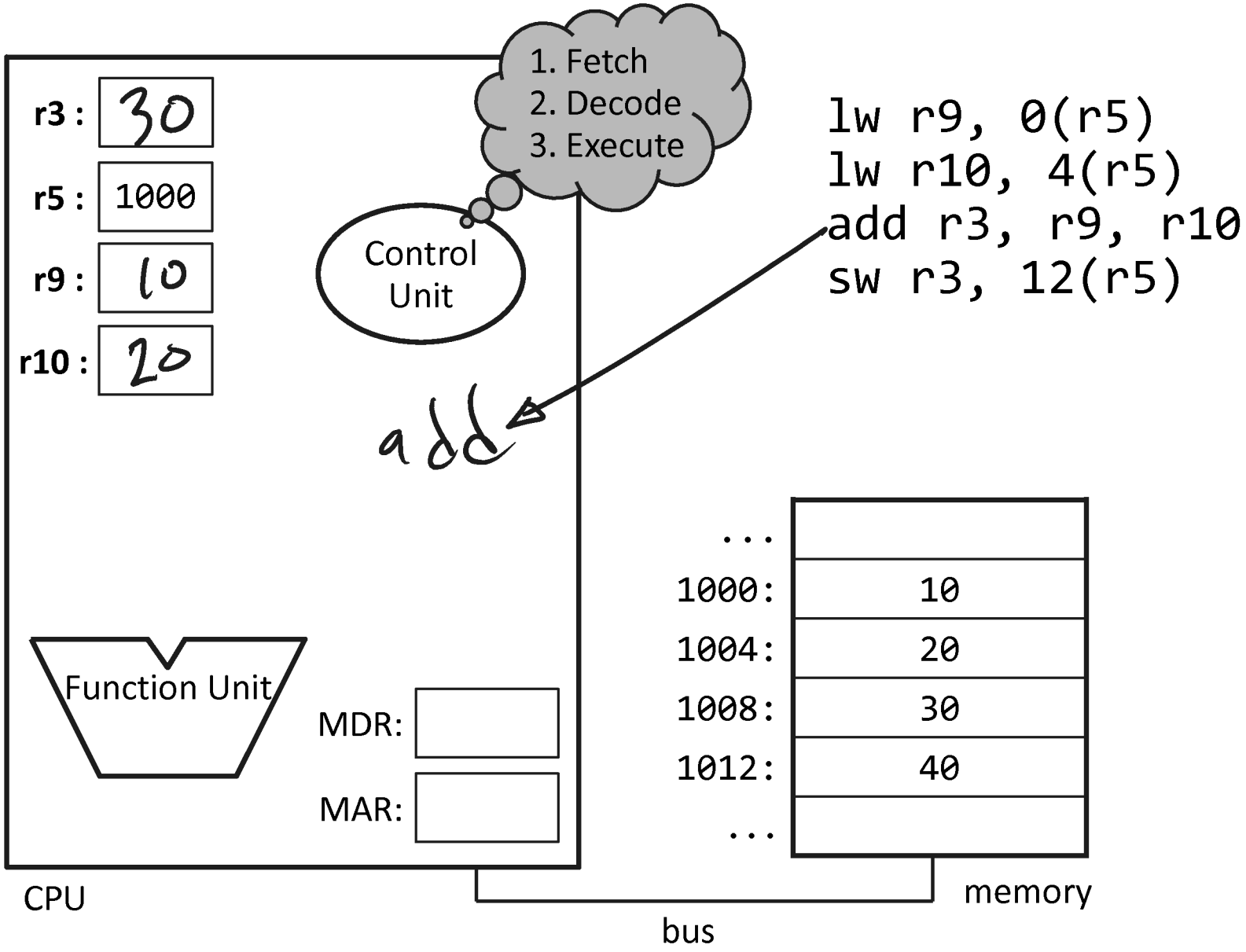

```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```

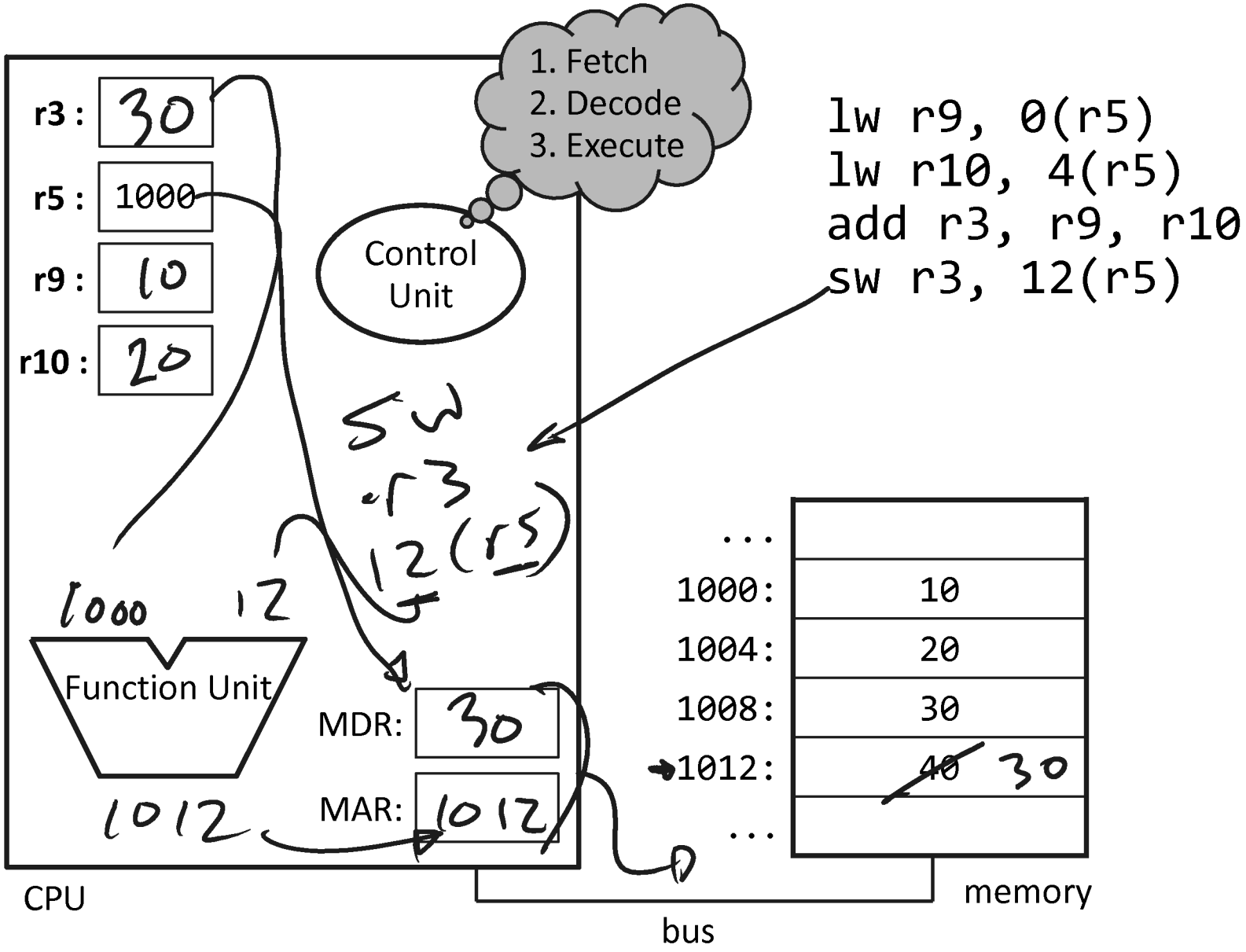



```
addi r5, r0, 10
muli r5, r5, 2
addi r5, r5, 15
```


1. Fetch
2. Decode
3. Execute
lw r9, 0(r5)
lw r10, 4(r5)
add r3, r9, r10
sw r3, 12(r5)
r3 :
r5 : 1000
r9 : 10
r10 :
Control Unit
lw
r9
0(r5)
1000
0
Function Unit
+
1000
MDR: 10
MAR: 1000
...
1000: 10
1004: 20
1008: 30
1012: 40
...
CPU
bus
memory


1. Fetch
2. Decode
3. Execute
lw r9, 0(r5)
lw r10, 4(r5)
add r3, r9, r10
sw r3, 12(r5)
r3 :
r5 : 1000
r9 : 10
r10 : 20
Control Unit
lw
r10
4(r5)
1000
4
Function Unit
+
1004
MDR: 20
MAR: 1004
...
1000: 10
1004: 20
1008: 30
1012: 40
...
CPU
memory
bus


1. Fetch
2. Decode
3. Execute
Control Unit
r3 : 30
r5 : 1000
r9 : 10
r10 : 20
add
lw r9, 0(r5)
lw r10, 4(r5)
add r3, r9, r10
sw r3, 12(r5)
Function Unit
MDR:
MAR:
CPU
...
1000: 10
1004: 20
1008: 30
1012: 40
...
memory
bus


1. Fetch
2. Decode
3. Execute
r3 : 30
r5 : 1000
r9 : 10
r10 : 20
Control Unit
lw r9, 0(r5)
lw r10, 4(r5)
add r3, r9, r10
sw r3, 12(r5)
sw
r3
12(r5)
1000
12
Function Unit
MDR: 30
MAR: 1012
1012
...
1000: 10
1004: 20
1008: 30
1012: 40 30
...
CPU
bus
memory

Machine language represents program as numbers

- Store in / fetch from memory like other data
- 2 new registers:
  - Program counter (PC): address of next instruction
  - Instruction register (IR): current instruction

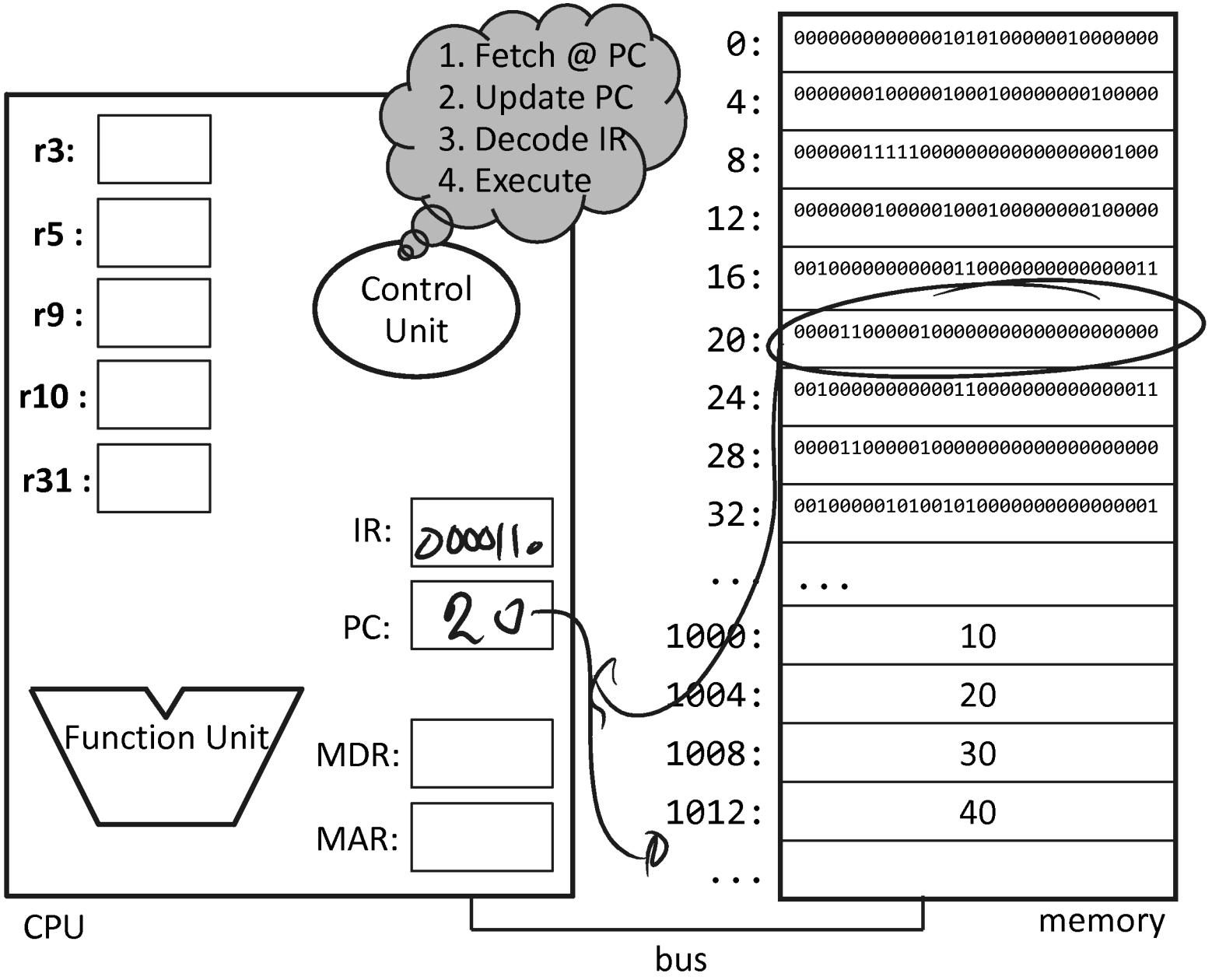
Revolutionary idea: a program is *just data*

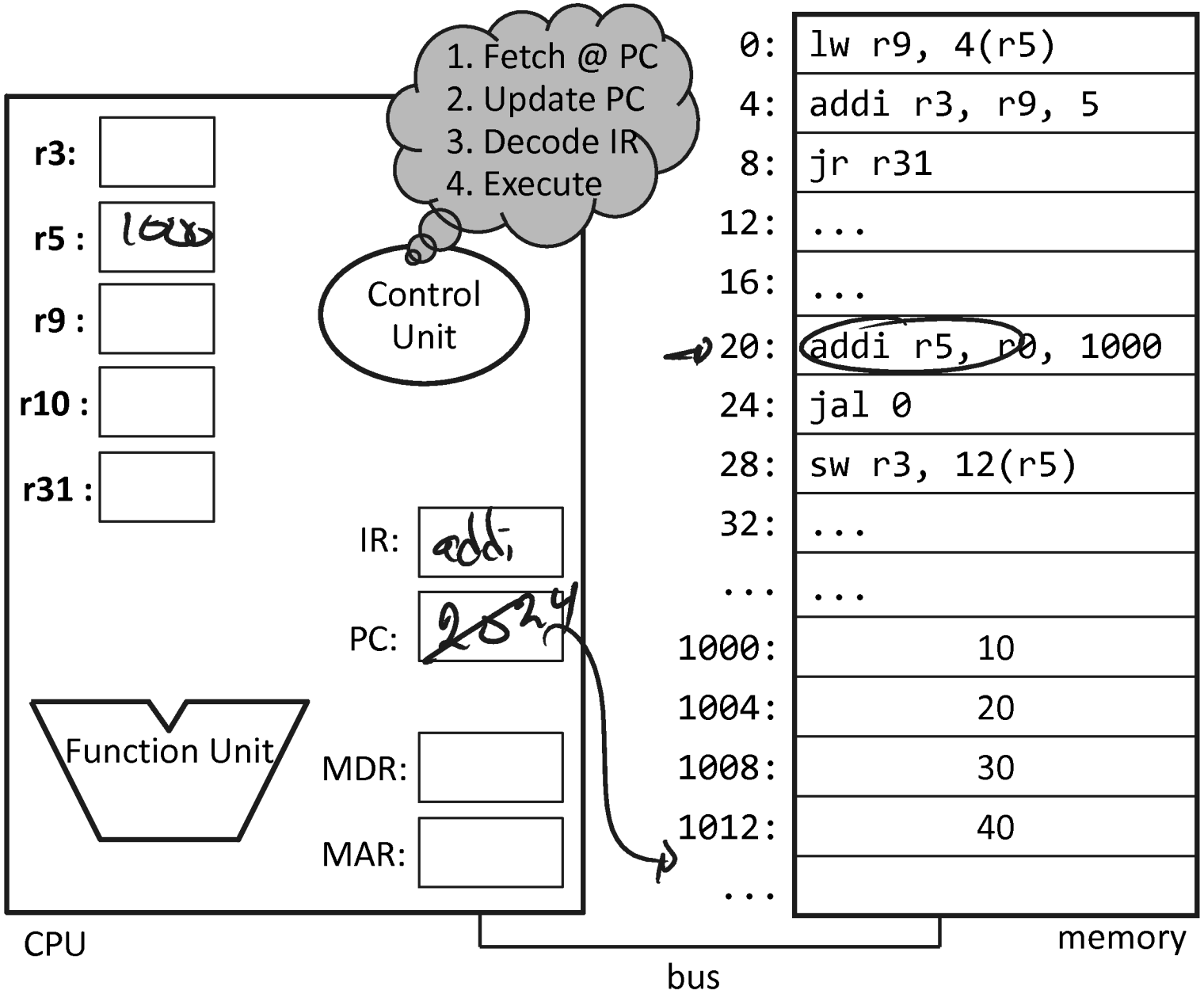
→ von Neumann Architecture

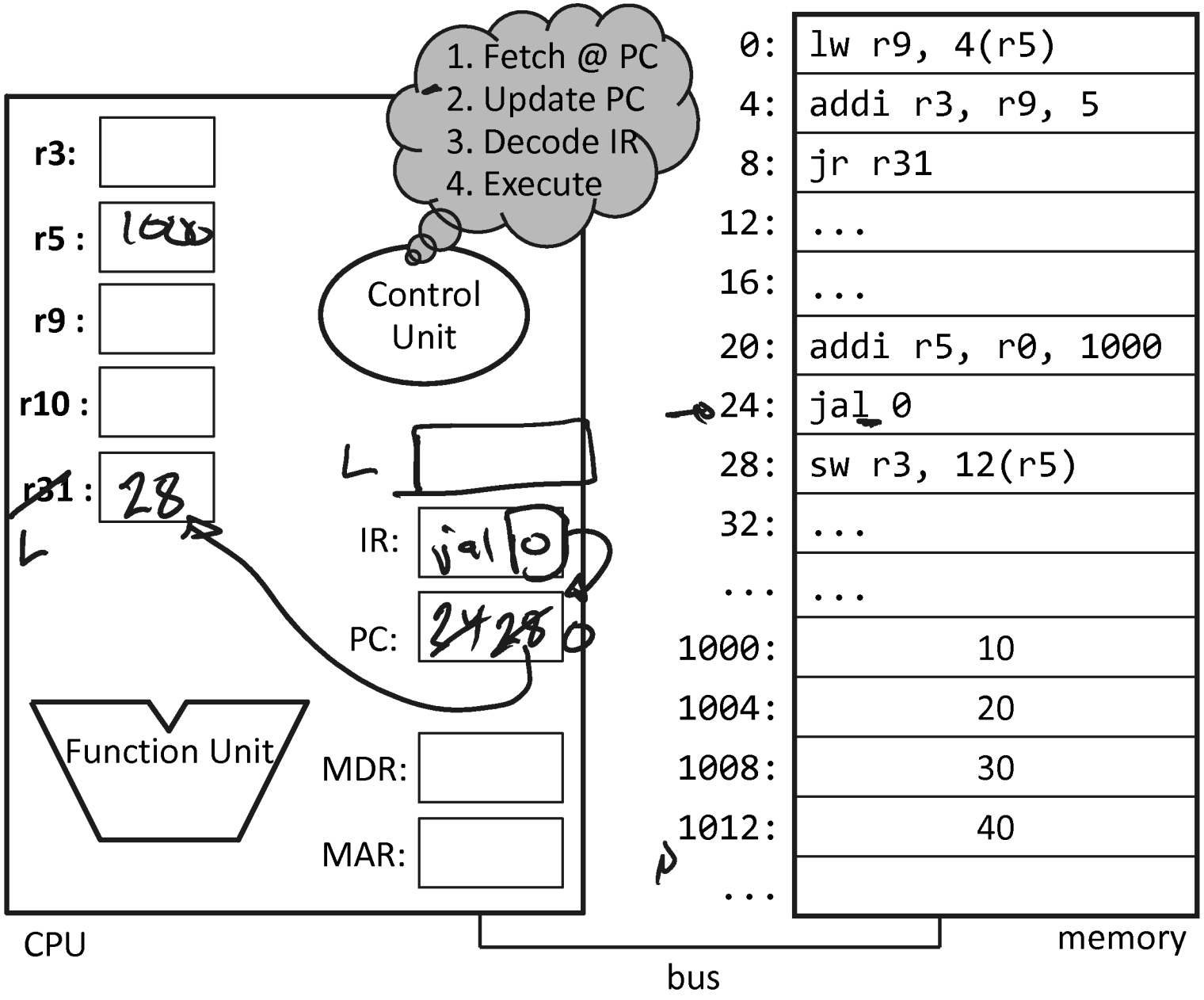
Alternative:

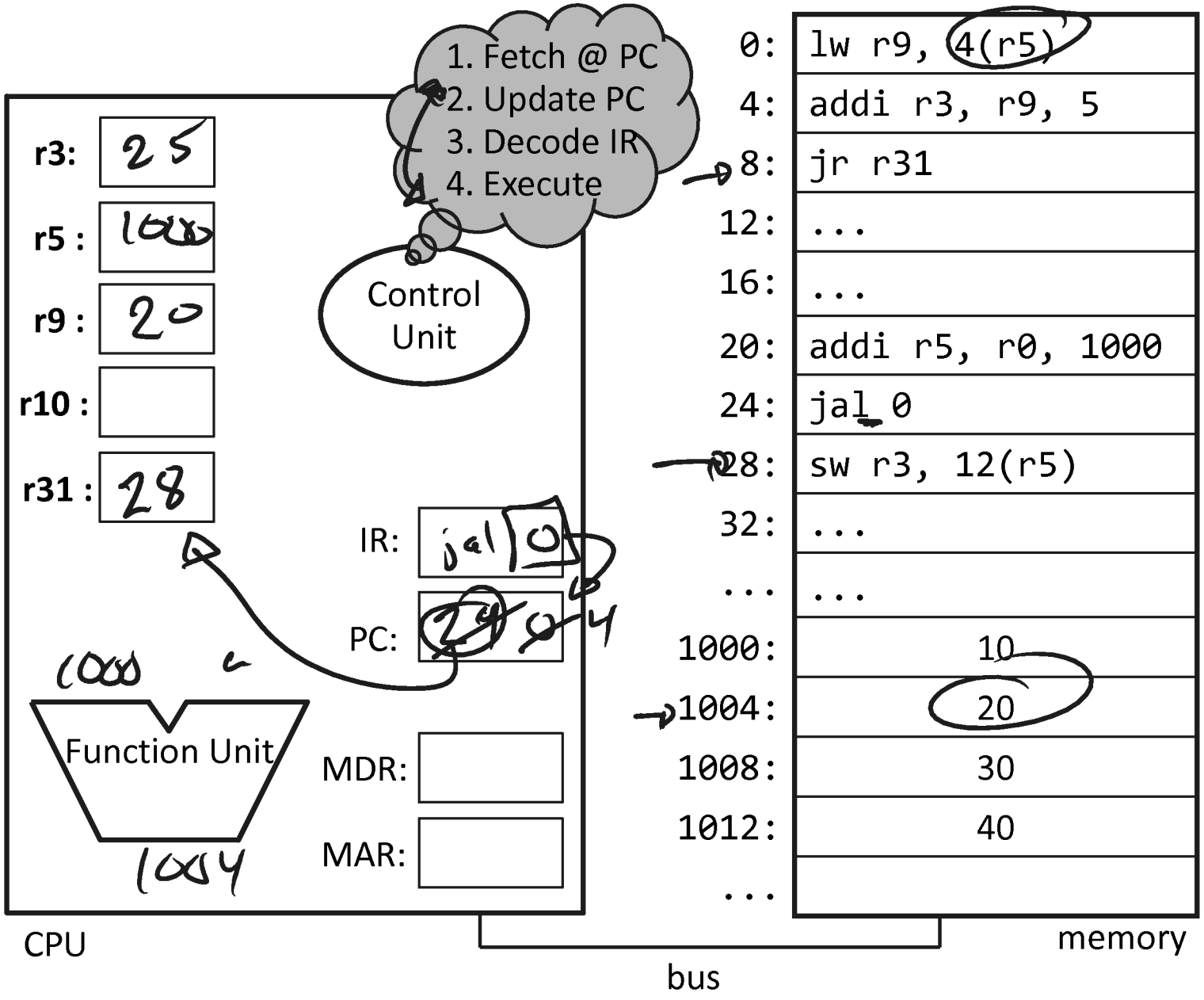
- Separate memory systems for code and data

→ Harvard Architecture


1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute
Control Unit
r3:
r5 :
r9 :
r10 :
r31 :
IR: 000110
PC: 20
Function Unit
MDR:
MAR:
CPU
bus
memory
0: 00000000000001010100000010000000
4: 00000001000001000100000000100000
8: 00000011111000000000000000001000
12: 00000001000001000100000000100000
16: 00100000000000110000000000000011
20: 00001100000100000000000000000000
24: 00100000000000110000000000000011
28: 00001100000100000000000000000000
32: 00100000101001010000000000000001
...
1000: 10
1004: 20
1008: 30
1012: 40


1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute
Control Unit
r3:
r5 :
r9 :
r10 :
r31 :
IR:
PC:
Function Unit
MDR:
MAR:
CPU
bus
memory
0: lw r9, 4(r5)
4: addi r3, r9, 5
8: jr r31
12: ...
16: ...
20: addi r5, r0, 1000
24: jal 0
28: sw r3, 12(r5)
32: ...
... ...
1000: 10
1004: 20
1008: 30
1012: 40
...

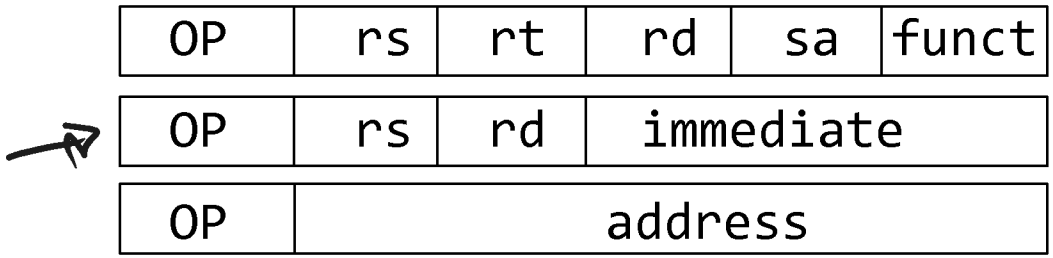

1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute
Control Unit
r3:
r5 :
r9 :
r10 :
r31 :
28
IR:
jal 0
PC:
Function Unit
MDR:
MAR:
CPU
0: lw r9, 4(r5)
4: addi r3, r9, 5
8: jr r31
12: ...
16: ...
20: addi r5, r0, 1000
24: jal 0
28: sw r3, 12(r5)
32: ...
... ...
1000: 10
1004: 20
1008: 30
1012: 40
...
memory
bus

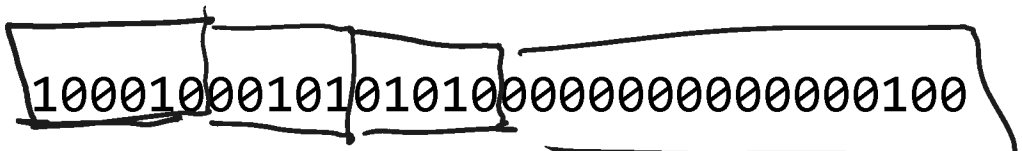

1. Fetch @ PC
2. Update PC
3. Decode IR
4. Execute
Control Unit
r3: 25
r5: 1000
r9: 20
r10:
r31: 28
IR: jal 0
PC: 24 28
1000
Function Unit
1004
MDR:
MAR:
CPU
0: lw r9, 4(r5)
4: addi r3, r9, 5
8: jr r31
12: ...
16: ...
20: addi r5, r0, 1000
24: jal 0
28: sw r3, 12(r5)
32: ...
... ...
1000: 10
1004: 20
1008: 30
1012: 40
...
memory
bus

MIPS R3000 ISA (Instruction Set Architecture)
Interface between hardware and software

- memory: load, store, ...
- computational: add, sub, mul, ...
- control: jump, branch, ...
- floating point, cpu and memory management, ...

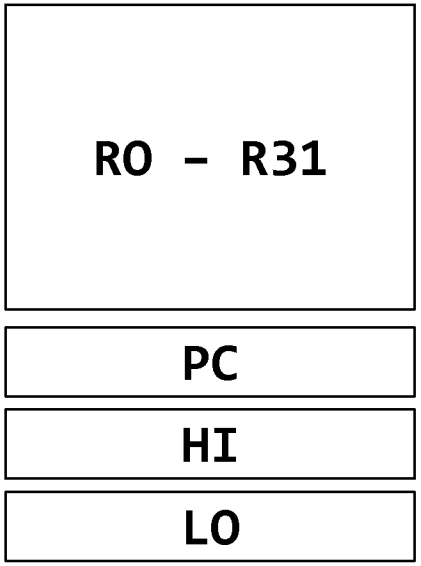
Instruction Formats

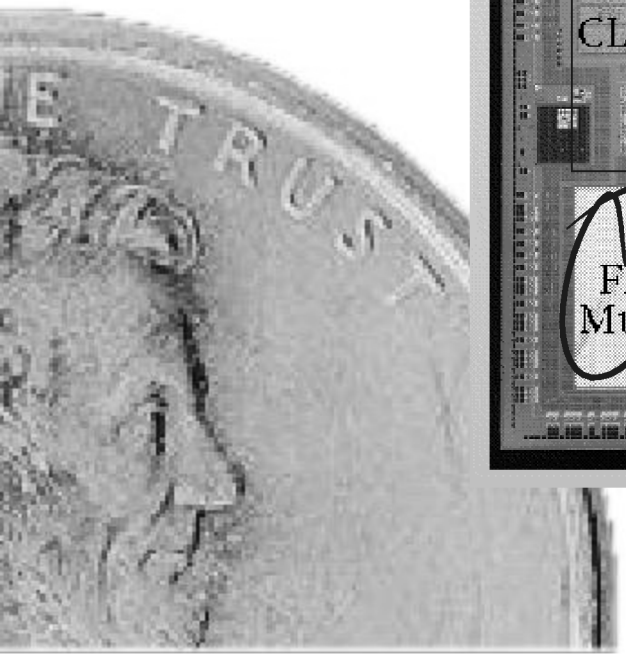




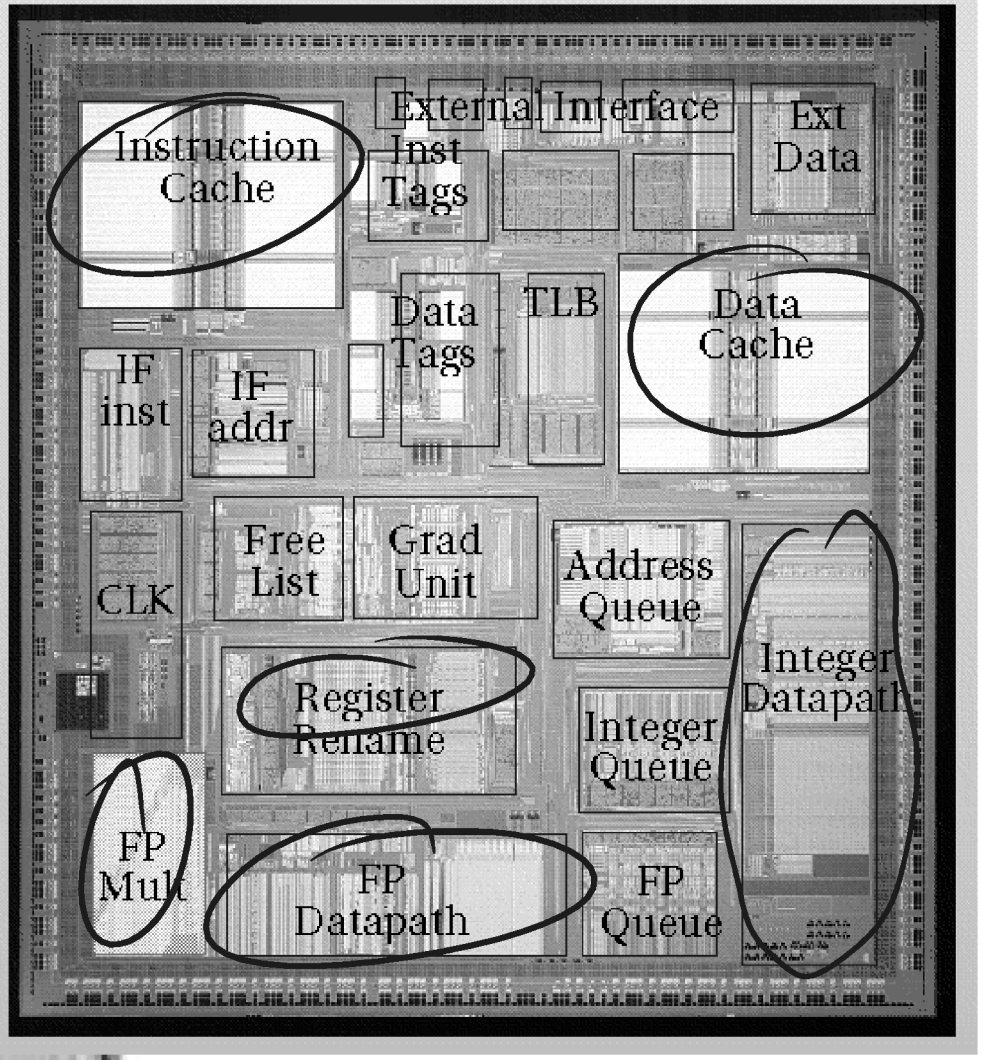

```
lw r10, 4(r5)
```

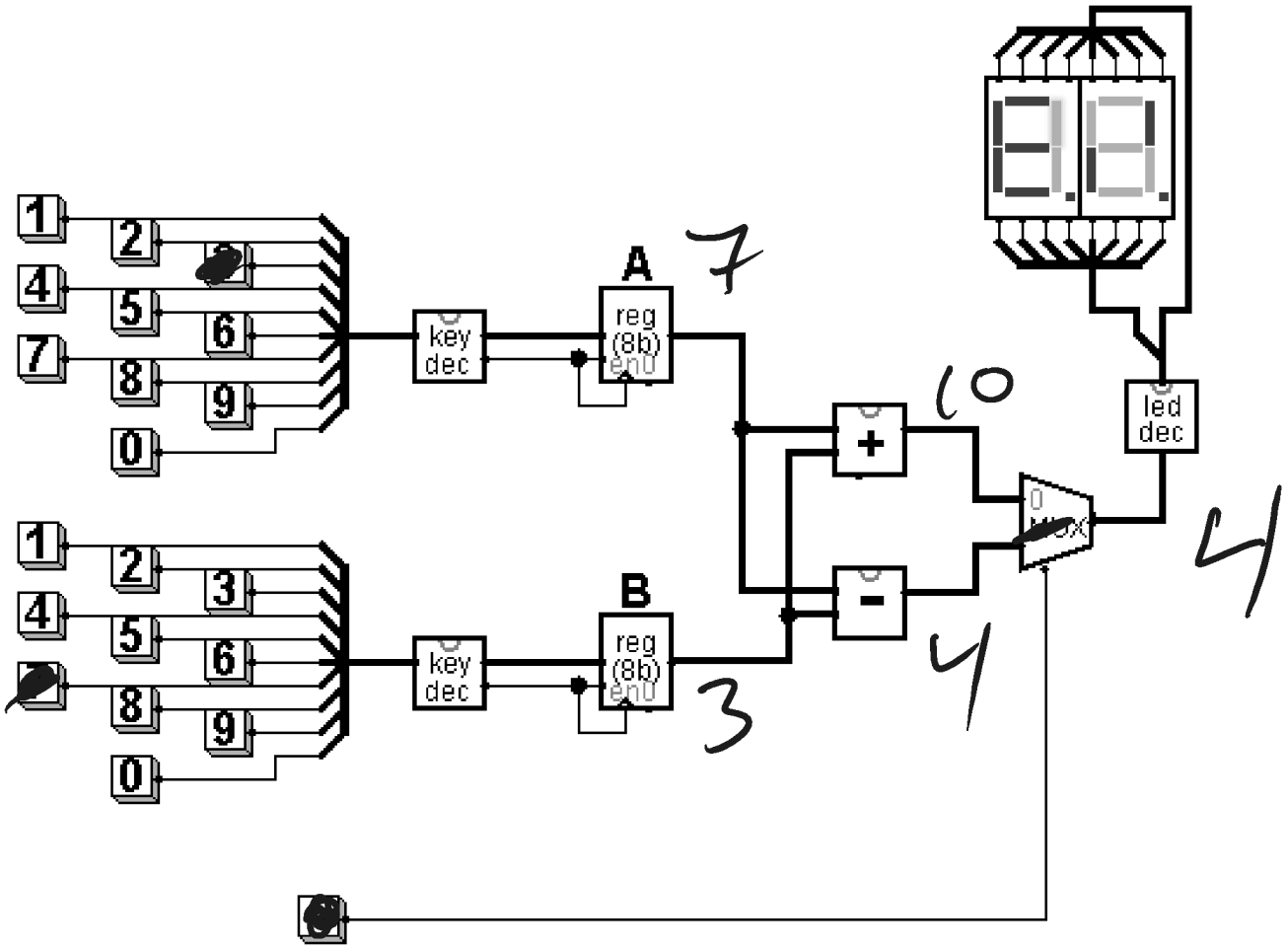
Registers



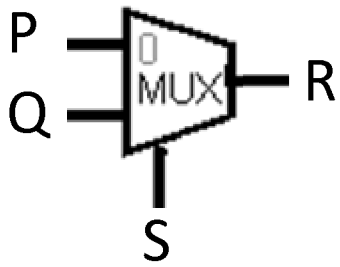


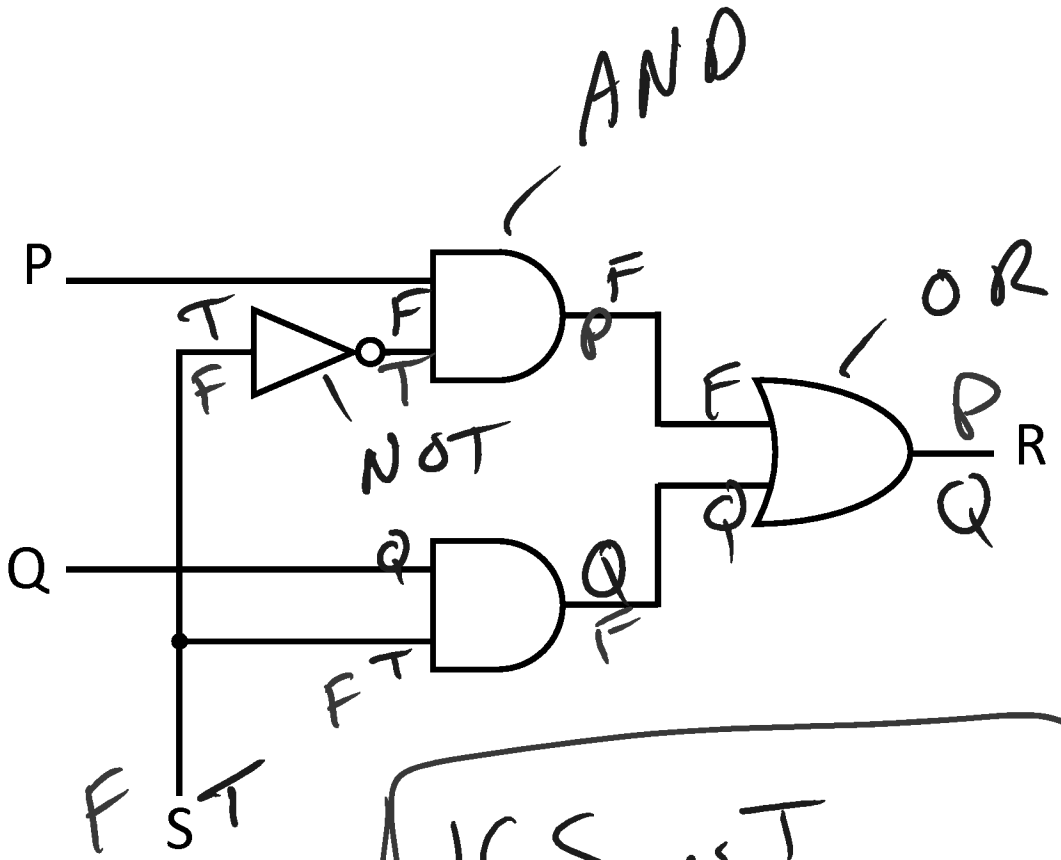
R10K die size 16.6mm x 17.9mm




1
2
4
5
6
7
8
9
0
key dec
A
reg (8b)
en
7
+
10
led dec
1
2
3
4
5
6
8
9
0
key dec
B
reg (8b)
en
3
-
4
4

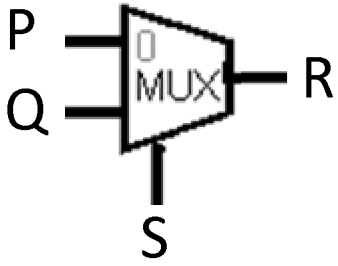
1-bit Multiplexor



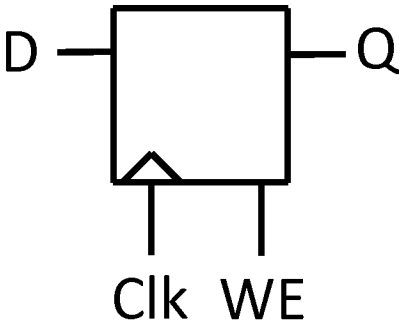


If S is T
then R = Q
If S is F
then R = P

Computation
E.g. Multiplexor

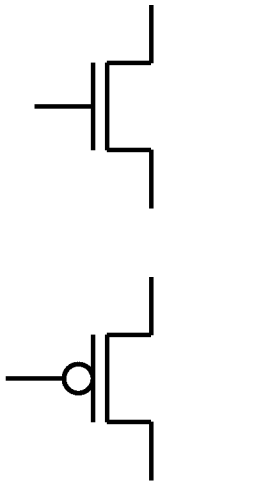


State
E.g. Register



Gates
E.g. AND

Transistors



Why?

```
void A() {
    for (int i = 0; i < 4096; i++)
        for (int j = 0; j < 4096; j++)
            v[i][j] = f(v[i][j], i, j);
}
```

0.45 sec, (0.12 sec optimized)

```
void B() {
    for (int j = 0; j < 4096; j++)
        for (int i = 0; i < 4096; i++)
            v[i][j] = f(v[i][j], i, j);
}
```

4.05 sec (3.52 sec optimized)

The number of transistors integrated on a single die will double every 24 months...

– Gordon Moore, Intel co-founder, 1965

1971 – 2300 transistors – 1MHz – 4004

1990 – 1M transistors – 50MHz – i486

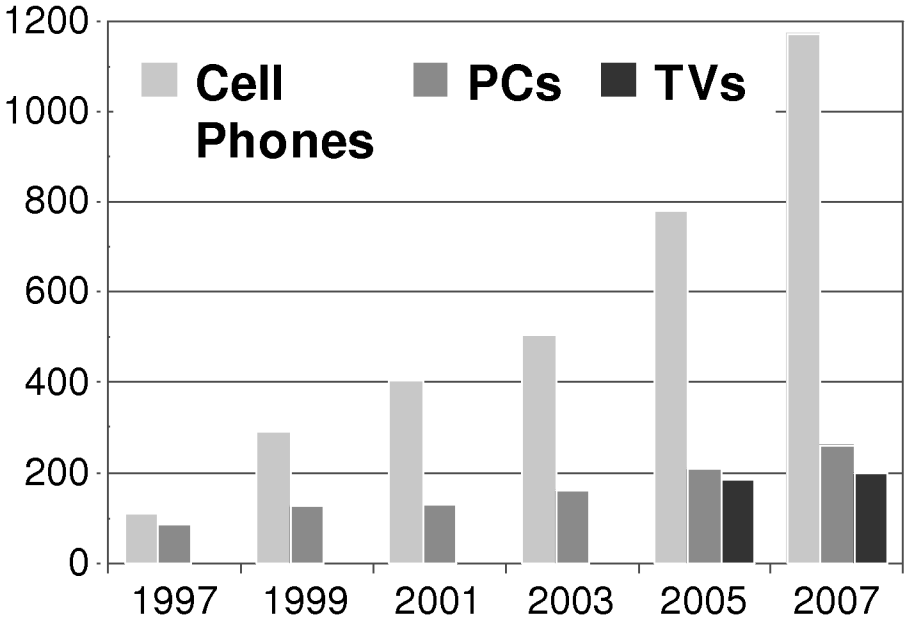
2001 – 42M transistors – 2GHz – Xeon

2004 – 55M transistors – 3GHz – P4

2007 – 290M transistors – 3GHz – Core 2 Duo

2009 – 731M transistors – 2GHz – Nehalem

Xilinx FPGA



NVidia GPU

Berkeley mote

Why?

- Basic knowledge needed for *all* other areas of CS:
  operating systems, compilers, ...
- Levels are not independent
  hardware design ↔ software design ↔ performance
- Crossing boundaries is hard but important
  device drivers
- Good design techniques
  abstraction, layering, pipelining, parallel vs. serial, ...
- Understand where the world is going

http://www.cs.cornell.edu/courses/cs3410

- Office Hours / Consulting Hours
- Lecture slides & schedule
- Logisim
- CSUG lab access (esp. second half of course)

Sections (choose one):

T	2:55 – 4:10pm	Hollister 110
W	3:35 – 4:50pm	Hollister 320
R	11:40 – 12:55pm	Hollister 401
R	2:55 – 4:10pm	Hollister 401
F	2:55 – 4:10pm	Snee 1150

- Will cover new material
- This week: intro to logisim

A) Love it
B) Okay
C) What?
D) Whatever
E) Please don't

Grading:

- 4 Programming Assignments (35 – 45%)
  - Work in groups of two
- 2 Prelims (30 – 40%)
- 4-5 Homework Assignments (20 – 25%)
  - Work alone
- Discretionary (5%)

Academic Integrity:

- All submitted work must be your own (or your groups)
  - OK to study together, but do not share solutions
- Cite your sources

Stressed? Tempted? Lost?

- Come see me before due date!

Plagiarism in any form will not be tolerated