

The TOY Machine



Introduction to Computer Science • Robert Sedgewick and Kevin Wayne • Copyright © 2005 • <http://www.cs.Princeton.EDU/IntroCS>

Basic Characteristics of TOY Machine

TOY is a general-purpose computer.

- Sufficient power to perform ANY computation.
- Limited only by amount of memory and time.



John von Neumann

Stored-program computer. (von Neumann memo, 1944)

- Data and instructions encoded in binary.
- Data and instructions stored in SAME memory.



Maurice Wilkes (left)
EDSAC (right)

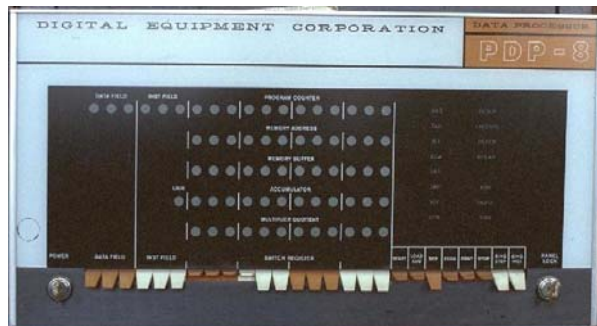
All modern computers are general-purpose computers and have same (von Neumann/Princeton) architecture.

2

What is TOY?

An imaginary machine similar to:

- Ancient computers. (PDP-8, world's first commercially successful minicomputer. 1960s)
 - 12-bit words
 - 2K words of memory
 - Used in Apollo project

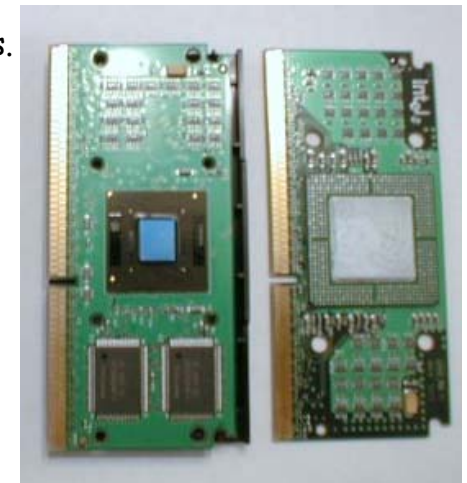


3

What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.



Pentium

Celeron

4

What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.

Why study TOY?

- Machine language programming.
 - how do high-level programs relate to computer?
 - a favor of assembly programming
- Computer architecture.
 - how is a computer put together?
 - how does it work?
- Optimized for understandability, not cost or performance.

5

Inside the Box

Switches. Input data and programs.

Lights. View data.

Memory.

- Stores data and programs.
- 256 "words." (16 bits each)
- Special word for stdin / stdout.

Program counter (PC).

- An extra 8-bit register.
- Keeps track of next instruction to be executed.

Registers.

- Fastest form of storage.
- Scratch space during computation.
- 16 registers. (16 bits each)
- Register 0 is always 0.

Arithmetic-logic unit (ALU). Manipulate data stored in registers.

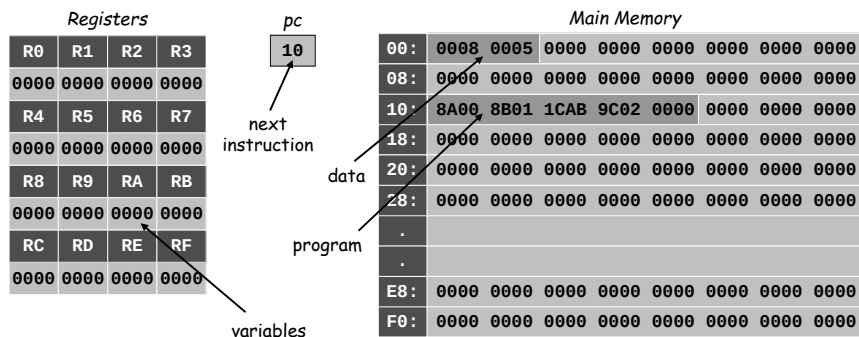
Standard input, standard output. Interact with outside world.

6

Machine "Core" Dump

Machine contents at a particular place and time.

- Record of what program has done.
- Completely determines what machine will do.



7

Program and Data

Program: Sequence of instructions.

16 instruction types:

- 16-bit word (interpreted one way).
- Changes contents of registers, memory, and PC in specified, well-defined ways.

Data:

- 16-bit word (interpreted other way).

Program counter (PC):

- Stores memory address of "next instruction."
- TOY usually starts at address 10.

Instructions

0:	halt
1:	add
2:	subtract
3:	and
4:	xor
5:	shift left
6:	shift right
7:	load address
8:	load
9:	store
A:	load indirect
B:	store indirect
C:	branch zero
D:	branch positive
E:	jump register
F:	jump and link

8

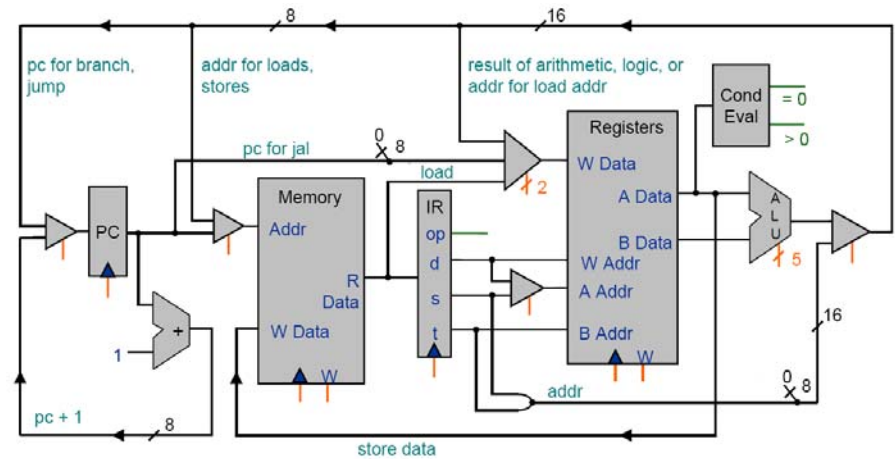
TOY Reference Card

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

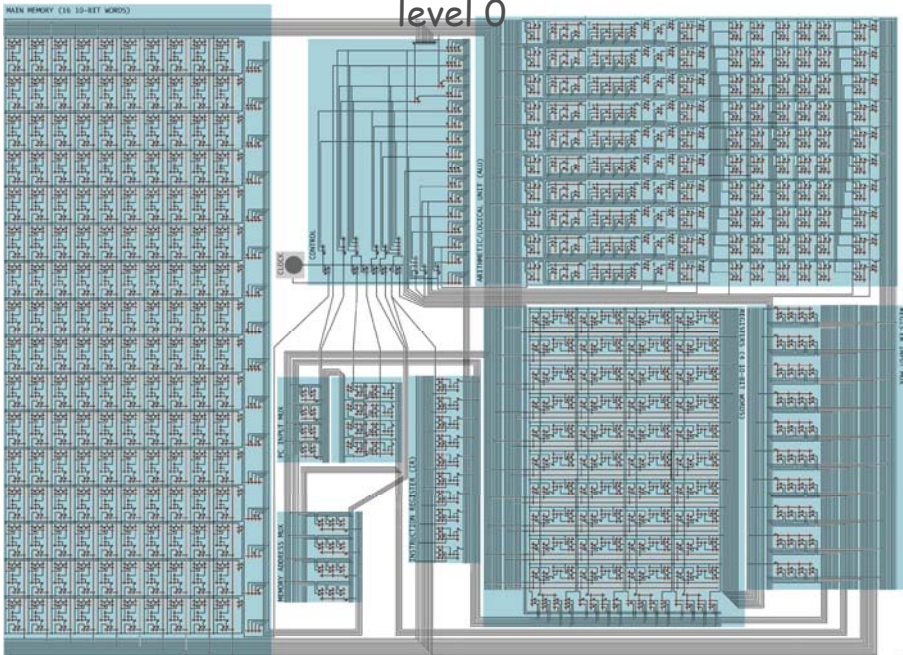
#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	$R[d] \leftarrow R[s] + R[t]$
2:	subtract	1	$R[d] \leftarrow R[s] - R[t]$
3:	and	1	$R[d] \leftarrow R[s] \& R[t]$
4:	xor	1	$R[d] \leftarrow R[s] \wedge R[t]$
5:	shift left	1	$R[d] \leftarrow R[s] \ll R[t]$
6:	shift right	1	$R[d] \leftarrow R[s] \gg R[t]$
7:	load addr	2	$R[d] \leftarrow \text{addr}$
8:	load	2	$R[d] \leftarrow \text{mem}[\text{addr}]$
9:	store	2	$\text{mem}[\text{addr}] \leftarrow R[d]$
A:	load indirect	1	$R[d] \leftarrow \text{mem}[R[t]]$
B:	store indirect	1	$\text{mem}[R[t]] \leftarrow R[d]$
C:	branch zero	2	if $(R[d] == 0)$ pc $\leftarrow$ addr
D:	branch positive	2	if $(R[d] > 0)$ pc $\leftarrow$ addr
E:	jump register	1	pc $\leftarrow R[t]$
F:	jump and link	2	$R[d] \leftarrow$ pc; pc $\leftarrow$ addr

Register 0 always 0.
Loads from mem[FF]
from stdin.
Stores to mem[FF] to
stdout.

TOY Architecture (level 1)



level 0



Programming in TOY

• Adds $8 + 5 = D$.

A Sample Program

A sample program.

- Adds $8 + 5 = D$.

RA	RB	RC	pc
0000	0000	0000	10

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

Memory

Since PC = 10, machine interprets 8A00 as an instruction.

13

Load

Load. (opcode 8)

- Loads the contents of some memory location into a register.
- 8A00 means load the contents of memory cell 00 into register A.

RA	RB	RC	pc
0000	0000	0000	10

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
8 ₁₆				A ₁₆				00 ₁₆							
opcode				dest d				addr							

14

Load

Load. (opcode 8)

- Loads the contents of some memory location into a register.
- 8B01 means load the contents of memory cell 01 into register B.

RA	RB	RC	pc
0008	0000	0000	11

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
8 ₁₆				B ₁₆				01 ₁₆							
opcode				dest d				addr							

15

Add

Add. (opcode 1)

- Add contents of two registers and store sum in a third.
- 1CAB adds the contents of registers A and B and put the result into register C.

RA	RB	RC	pc
0008	0005	0000	12

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1 ₁₆				C ₁₆				A ₁₆				B ₁₆			
opcode				dest d				source s				source t			

16

Store

Store. (opcode 9)

- Stores the contents of some register into a memory cell.
- 9CFF means store the contents of register c into memory cell FF (stdout).

RA	RB	RC	pc
0008	0005	000D	13

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	0	1	1	1	1	1	1	1	1
9 ₁₆				C ₁₆				FF ₁₆							
opcode				dest d				addr							

17

Halt

Halt. (opcode 0)

- Stop the machine.

RA	RB	RC	pc
0008	0005	000D	14

Registers

00: 0008	8	add.toy
01: 0005	5	
10: 8A00	RA $\leftarrow$ mem[00]	
11: 8B01	RB $\leftarrow$ mem[01]	
12: 1CAB	RC $\leftarrow$ RA + RB	
13: 9CFF	mem[FF] $\leftarrow$ RC	
14: 0000	halt	

18

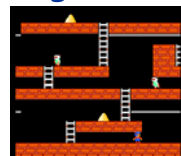
Simulation

Consequences of simulation.

- Test out new machine or microprocessor using simulator.
 - cheaper and faster than building actual machine
- Easy to add new functionality to simulator.
 - trace, single-step, breakpoint debugging
 - simulator more useful than TOY itself
- Reuse software from old machines.

Ancient programs still running on modern computers.

- Lode Runner on Apple IIe.
- Gameboy simulator on PCs.



19

Interfacing with the TOY Machine

To enter a program or data:

- Set 8 memory address switches.
- Set 16 data switches.
- Press LOAD.
 - data written into addressed word of memory

To view the results of a program:

- Set 8 memory address switches.
- Press LOOK: contents of addressed word appears in lights.



20

Using the TOY Machine: Run

To run the program:

- Set 8 memory address switches to address of first instruction.
- Press LOOK to set PC to first instruction.
- Press RUN button to repeat fetch-execute cycle until halt opcode.



21

Branch in TOY

To harness the power of TOY, need loops and conditionals.

- Manipulate PC to control program flow.

Branch if zero. (opcode C)

- Changes PC depending of value of some register.
- Used to implement: for, while, if-else.

Branch if positive. (opcode D)

- Analogous.

22

An Example: Multiplication

Multiply.

- No direct support in TOY hardware.
- Load in integers a and b, and store $c = a \times b$.
- Brute-force algorithm:
 - initialize $c = 0$
 - add b to c, a times

```
int a = 3;
int b = 9;
int c = 0;

while (a != 0) {
    c = c + b;
    a = a - 1;
}
```

Java

Issues ignored: slow, overflow, negative numbers.

23

Multiply

```
int a = 3;
int b = 9;
int c = 0;

while (a != 0) {
    c = c + b;
    a = a - 1;
}
```

24

Multiply

0A: 0003 3 ← inputs
 0B: 0009 9
 0C: 0000 0 ← output
 0D: 0000 0 ← constants
 0E: 0001 1

10: 8A0A RA ← mem[0A] a
 11: 8B0B RB ← mem[0B] b
 12: 8C0D RC ← mem[0D] c = 0
 13: 810E R1 ← mem[0E] always 1
 14: CA18 if (RA == 0) pc ← 18 while (a != 0) {
 15: 1CCB RC ← RC + RB c = c + b
 16: 2AA1 RA ← RA - R1 a = a - 1
 17: C014 pc ← 14 }
 18: 9CFF mem[FF] ← RC
 19: 0000 halt

loop

multiply.toy

25

Step-By-Step Trace

		R1	RA	RB	RC
10: 8A0A	RA ← mem[0A]		0003		
11: 8B0B	RB ← mem[0B]			0009	
12: 8C0D	RC ← mem[0D]				0000
13: 810E	R1 ← mem[0E]	0001			
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0009
16: 2AA1	RA ← RA - R1		0002		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0012
16: 2AA1	RA ← RA - R1		0001		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				001B
16: 2AA1	RA ← RA - R1		0000		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
18: 9CFF	mem[FF] ← RC				
19: 0000	halt				

multiply.toy

26

An Efficient Multiplication Algorithm

Inefficient multiply.

- Brute force multiplication algorithm loops a times.
- In worst case, 65,535 additions!

"Grade-school" multiplication.

- Always 16 additions to multiply 16-bit integers.

Decimal

1	2	3	4	
*	1	5	1	2
2 4 6 8				
1 2 3 4				
6 1 7 0				
1 2 3 4				
0 1 8 6 5 8 0 8				

Binary

1	0	1	1	
*	1	1	0	1
1 0 1 1				
0 0 0 0				
1 0 1 1				
1 0 1 1				
1 0 0 0 1 1 1 1				

27

Binary Multiplication

Grade school binary multiplication algorithm to compute $c = a \times b$.

- Initialize $c = 0$.
- Loop over i bits of b .
 - if $b_i = 0$, do nothing ← $b_i = i^{\text{th}}$ bit of b
 - if $b_i = 1$, shift a left i bits and add to c

1	0	1	1	a
*	1	1	0	b
1 0 1 1				
0 0 0 0				
1 0 1 1				
1 0 1 1				
1 0 0 0 1 1 1 1				
c				

Implement with built-in TOY shift instructions.

```
int c = 0;
for (int i = 15; i >= 0; i--)
  if (((b >> i) & 1) == 1)
    c = c + (a << i);
```

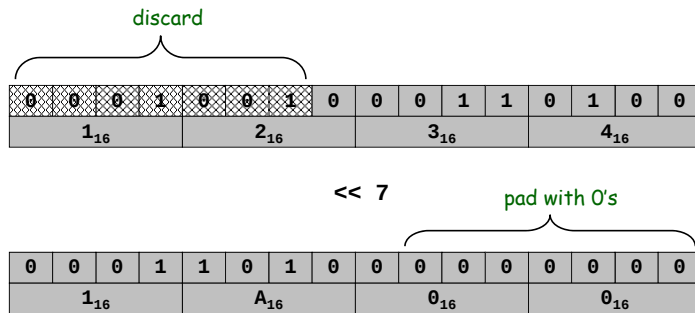
← $b_i = i^{\text{th}}$ bit of b

28

Shift Left

Shift left. (opcode 5)

- Move bits to the left, padding with zeros as needed.
- $1234_{16} \ll 7 = 1A00_{16}$

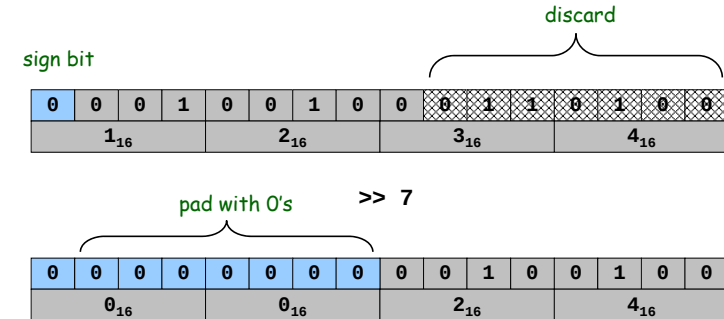


29

Shift Right

Shift right. (opcode 6)

- Move bits to the right, padding with sign bit as needed.
- $1234_{16} \gg 7 = 0024_{16}$

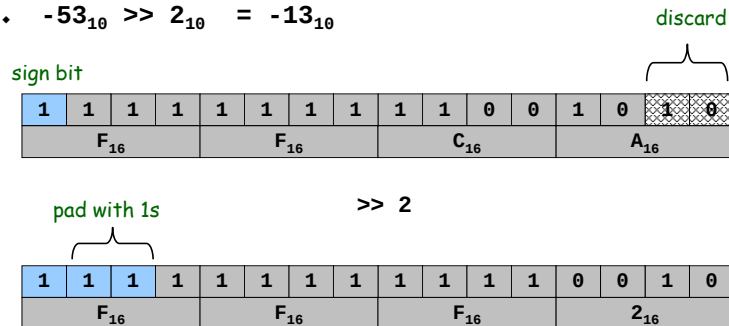


30

Shift Right (Sign Extension)

Shift right. (opcode 6)

- Move bits to the right, padding with sign bit as needed.
- $FFCA_{16} \gg 2 = FFF2_{16}$
- $-53_{10} \gg 2_{10} = -13_{10}$



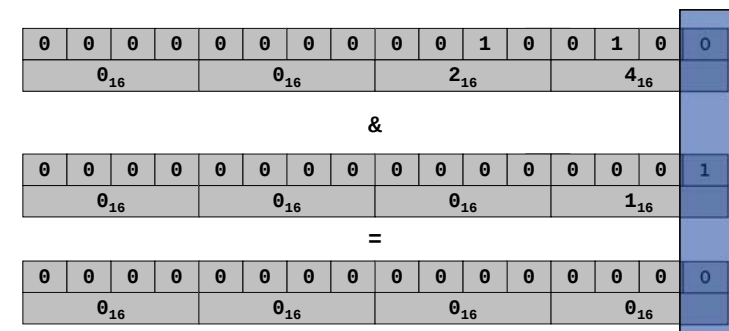
31

Bitwise AND

Logical AND. (opcode 3)

- Logic operations are BITWISE.
- $0024_{16} \& 0001_{16} = 0000_{16}$

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1



32

Shifting and Masking

Shift and mask: get the 7th bit of 1234.

- Compute $1234_{16} \gg 7_{16} = 0024_{16}$.
- Compute $0024_{16} \& 1_{16} = 0_{16}$.

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
1 ₁₆				2 ₁₆				3 ₁₆				4 ₁₆			
>> 7															
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
0 ₁₆				0 ₁₆				2 ₁₆				4 ₁₆			
&															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0 ₁₆				0 ₁₆				0 ₁₆				1 ₁₆			
=															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 ₁₆				0 ₁₆				0 ₁₆				0 ₁₆			

33

Binary Multiplication

```
int c = 0;
for (int i = 15; i >= 0; i--)
    if (((b >> i) & 1) == 1)
        c = c + (a << i);
```

34

Binary Multiplication

0A: 0003	3	←	inputs	
0B: 0009	9	←		
0C: 0000	0	←	output	
0D: 0000	0			
0E: 0001	1	←	constants	
0F: 0010	16			
10: 8A0A	RA ← mem[0A]		a	
11: 8B0B	RB ← mem[0B]		b	
12: 8C0D	RC ← mem[0D]		c = 0	
13: 810E	R1 ← mem[0E]		always 1	
14: 820F	R2 ← mem[0F]		i = 16 ← 16 bit words	
15: 2221	R2 ← R2 - R1		do {	
16: 53A2	R3 ← RA << R2		i--	
17: 64B2	R4 ← RB >> R2		a << i	
18: 3441	R4 ← R4 & R1		b >> i	
19: C41B	if (R4 == 0) goto 1B		b _i = i th bit of b	
1A: 1CC3	RC ← RC + R3		if b _i is 1	
1B: D215	if (R2 > 0) goto 15		add a << i to sum	
1C: 9CFF	mem[FF] ← RC		} while (i > 0);	

multiply-fast.toy

35

Useful TOY "Idioms"

Jump absolute.

- Jump to a fixed memory address.
 - branch if zero with destination
 - register 0 is always 0

```
17: C014  pc ← 14
```

Register assignment.

- No instruction that transfers contents of one register into another.
- Pseudo-instruction that simulates assignment:
 - add with register 0 as one of two source registers

```
17: 1230  R[2] ← R[3]
```

No-op.

- Instruction that does nothing.
- Plays the role of whitespace in C programs.
 - numerous other possibilities!

```
17: 1000  no-op
```

36

Standard Input and Output: Implications

Standard input and output enable you to:

- Process more information than fits in memory.
- Interact with the computer while it is running.

Standard output.

- Writing to memory location FF sends one word to TOY stdout.
- 9AFF writes the integer in register A to stdout.

Standard input.

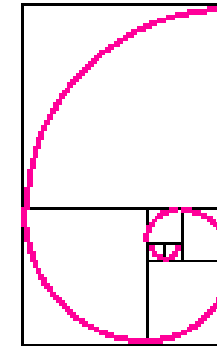
- Loading from memory address FF loads one word from TOY stdin.
- 8AFF reads in an integer from stdin and store it in register A.

37

Fibonacci Numbers

Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$



Reference: <http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibnat.html>

38

Standard Output

```
00: 0000 0
01: 0001 1

10: 8A00 RA ← mem[00]      a = 0
11: 8B01 RB ← mem[01]      b = 1
                               do {
12: 9AFF print RA           print a
13: 1AAB RA ← RA + RB       a = a + b
14: 2BAB RB ← RA - RB       b = a - b
15: DA12 if (RA > 0) goto 12 } while (a > 0)
16: 0000 halt
```

fibonacci.toy

```
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015
0016
0017
0018
0019
0020
0021
0022
0023
0024
0025
0026
0027
0028
0029
0030
0031
0032
0033
0034
0035
0036
0037
0038
0039
0040
0041
0042
0043
0044
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
0055
0056
0057
0058
0059
0060
0061
0062
0063
0064
0065
0066
0067
0068
0069
0070
0071
0072
0073
0074
0075
0076
0077
0078
0079
0080
0081
0082
0083
0084
0085
0086
0087
0088
0089
0090
0091
0092
0093
0094
0095
0096
0097
0098
0099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172
0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219
0220
0221
0222
0223
0224
0225
0226
0227
0228
0229
0230
0231
0232
0233
0234
0235
0236
0237
0238
0239
0240
0241
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251
0252
0253
0254
0255
0256
0257
0258
0259
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271
0272
0273
0274
0275
0276
0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349
0350
0351
0352
0353
0354
0355
0356
0357
0358
0359
0360
0361
0362
0363
0364
0365
0366
0367
0368
0369
0370
0371
0372
0373
0374
0375
0376
0377
0378
0379
0380
0381
0382
0383
0384
0385
0386
0387
0388
0389
0390
0391
0392
0393
0394
0395
0396
0397
0398
0399
0400
0401
0402
0403
0404
0405
0406
0407
0408
0409
0410
0411
0412
0413
0414
0415
0416
0417
0418
0419
0420
0421
0422
0423
0424
0425
0426
0427
0428
0429
0430
0431
0432
0433
0434
0435
0436
0437
0438
0439
0440
0441
0442
0443
0444
0445
0446
0447
0448
0449
0450
0451
0452
0453
0454
0455
0456
0457
0458
0459
0460
0461
0462
0463
0464
0465
0466
0467
0468
0469
0470
0471
0472
0473
0474
0475
0476
0477
0478
0479
0480
0481
0482
0483
0484
0485
0486
0487
0488
0489
0490
0491
0492
0493
0494
0495
0496
0497
0498
0499
0500
0501
0502
0503
0504
0505
0506
0507
0508
0509
0510
0511
0512
0513
0514
0515
0516
0517
0518
0519
0520
0521
0522
0523
0524
0525
0526
0527
0528
0529
0530
0531
0532
0533
0534
0535
0536
0537
0538
0539
0540
0541
0542
0543
0544
0545
0546
0547
0548
0549
0550
0551
0552
0553
0554
0555
0556
0557
0558
0559
0560
0561
0562
0563
0564
0565
0566
0567
0568
0569
0570
0571
0572
0573
0574
0575
0576
0577
0578
0579
0580
0581
0582
0583
0584
0585
0586
0587
0588
0589
0590
0591
0592
0593
0594
0595
0596
0597
0598
0599
0600
0601
0602
0603
0604
0605
0606
0607
0608
0609
0610
0611
0612
0613
0614
0615
0616
0617
0618
0619
0620
0621
0622
0623
0624
0625
0626
0627
0628
0629
0630
0631
0632
0633
0634
0635
0636
0637
0638
0639
0640
0641
0642
0643
0644
0645
0646
0647
0648
0649
0650
0651
0652
0653
0654
0655
0656
0657
0658
0659
0660
0661
0662
0663
0664
0665
0666
0667
0668
0669
0670
0671
0672
0673
0674
0675
0676
0677
0678
0679
0680
0681
0682
0683
0684
0685
0686
0687
0688
0689
0690
0691
0692
0693
0694
0695
0696
0697
0698
0699
0700
0701
0702
0703
0704
0705
0706
0707
0708
0709
0710
0711
0712
0713
0714
0715
0716
0717
0718
0719
0720
0721
0722
0723
0724
0725
0726
0727
0728
0729
0730
0731
0732
0733
0734
0735
0736
0737
0738
0739
0740
0741
0742
0743
0744
0745
0746
0747
0748
0749
0750
0751
0752
0753
0754
0755
0756
0757
0758
0759
0760
0761
0762
0763
0764
0765
0766
0767
0768
0769
0770
0771
0772
0773
0774
0775
0776
0777
0778
0779
0780
0781
0782
0783
0784
0785
0786
0787
0788
0789
0790
0791
0792
0793
0794
0795
0796
0797
0798
0799
0800
0801
0802
0803
0804
0805
0806
0807
0808
0809
0810
0811
0812
0813
0814
0815
0816
0817
0818
0819
0820
0821
0822
0823
0824
0825
0826
0827
0828
0829
0830
0831
0832
0833
0834
0835
0836
0837
0838
0839
0840
0841
0842
0843
0844
0845
0846
0847
0848
0849
0850
0851
0852
0853
0854
0855
0856
0857
0858
0859
0860
0861
0862
0863
0864
0865
0866
0867
0868
0869
0870
0871
0872
0873
0874
0875
0876
0877
0878
0879
0880
0881
0882
0883
0884
0885
0886
0887
0888
0889
0890
0891
0892
0893
0894
0895
0896
0897
0898
0899
0900
0901
0902
0903
0904
0905
0906
0907
0908
0909
0910
0911
0912
0913
0914
0915
0916
0917
0918
0919
0920
0921
0922
0923
0924
0925
0926
0927
0928
0929
0930
0931
0932
0933
0934
0935
0936
0937
0938
0939
0940
0941
0942
0943
0944
0945
0946
0947
0948
0949
0950
0951
0952
0953
0954
0955
0956
0957
0958
0959
0960
0961
0962
0963
0964
0965
0966
0967
0968
0969
0970
0971
0972
0973
0974
0975
0976
0977
0978
0979
0980
0981
0982
0983
0984
0985
0986
0987
0988
0989
0990
0991
0992
0993
0994
0995
0996
0997
0998
0999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
```

Load Address (a.k.a. Load Constant)

Load address. (opcode 7)

- Loads an 8-bit integer into a register.
- 7A30 means load the value 30 into register A.

Applications.

- Load a small constant into a register.
- Load a 8-bit memory address into a register.
 - register stores "pointer" to a memory cell

`a = 30;`
Java code

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0	0	0	1	1	0	0	0	0
7 ₁₆				A ₁₆				3 ₁₆				0 ₁₆			
opcode				dest d				addr							

41

Arrays in TOY

TOY main memory is a giant array.

- Can access memory cell 30 using load and store.
- 8C30 means load mem[30] into register c.
- Goal: access memory cell i where i is a variable.

Load indirect. (opcode A)

- AC06 means load mem[R6] into register c.

Store indirect. (opcode B)

- BC06 means store contents of register c into mem[R6].

a variable index (like a pointer)
a variable index

```
for (int i = 0; i < N; i++)
    a[i] = StdIn.readInt();

for (int i = 0; i < N; i++)
    System.out.println(a[N-i-1]);
```

Reverse.java

42

TOY Implementation of Reverse

TOY implementation of reverse.

- Read in a sequence of integers and store in memory 30, 31, 32, ...
- Stop reading if 0000.
- Print sequence in reverse order.

TOY Implementation of Reverse

TOY implementation of reverse.

- Read in a sequence of integers and store in memory 30, 31, 32, ...
- Stop reading if 0000.
- Print sequence in reverse order.

```
10: 7101  R1 ← 0001          constant 1
11: 7A30  RA ← 0030          a[]
12: 7B00  RB ← 0000          n

13: 8CFF  read RC
14: CC19  if (RC == 0) goto 19
15: 16AB  R6 ← RA + RB
16: BC06  mem[R6] ← RC
17: 1BB1  RB ← RB + R1
18: C013  goto 13
```

read in the data

```
while(true) {
    c = StdIn.readInt();
    if (c == 0) break;
    address of a[n]
    a[n] = c;
    n++;
}
```

43

44

TOY Implementation of Reverse

TOY implementation of reverse.

- Read in a sequence of integers and store in memory 30, 31, 32, ...
- ➔ • Stop reading if 0000.
- Print sequence in reverse order.

```

19: CB20  if (RB == 0) goto 20      while (n > 0) {
1A: 16AB  R6 ← RA + RB              address of a[n]
1B: 2661  R6 ← R6 - R1              address of a[n-1]
1C: AC06  RC ← mem[R6]              c = a[n-1];
1D: 9CFF  write RC                  System.out.println(c);
1E: 2BB1  RB ← RB - R1              n--;
1F: C019  goto 19                  }
20: 0000  halt                      print in reverse order
    
```

45

Unsafe Code at any Speed

What happens if we make array start at 00 instead of 30?

- Self modifying program.
- Exploit buffer overrun and run arbitrary code!

```

10: 7101  R1 ← 0001                constant 1
11: 7A00  RA ← 0000                a[]
12: 7B00  RB ← 0000                n

13: 8CFF  read RC                  while(true) {
14: CC19  if (RC == 0) goto 19      c = StdIn.readInt();
15: 16AB  R6 ← RA + RB              if (c == 0) break;
16: BC06  mem[R6] ← RC              address of a[n]
17: 1BB1  RB ← RB + R1              a[n] = c;
18: C013  goto 13                  n++;
                                }
    
```

Crazy 8s Input

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
8	8	8	8	8	8	8	8
9	8	F	F	C	0	1	1

46

What Can Happen When We Lose Control?

Buffer overrun.

- Array buffer[] has size 100.
- User might enter 200 characters.
- Might lose control of machine behavior.
- Majority of viruses and worms caused by similar errors.

```

#include <stdio.h>
int main(void) {
    char buffer[100];
    scanf("%s", buffer);
    printf("%s\n", buffer);
    return 0;
}
    
```

unsafe C program

Robert Morris Internet Worm.

- Cornell grad student injected worm into Internet in 1988.
- Exploited buffer overrun in finger daemon fingerd.

47

Function Call: A Failed Attempt

Goal: $x \times y \times z$.

- Need two multiplications: $x \times y$, $(x \times y) \times z$.
- Solution 1: write multiply code 2 times.
- Solution 2: write a TOY function.

A failed attempt:

- Write multiply loop at 30-36.
- Calling program agrees to store arguments in registers A and B.
- Function agrees to leave result in register C.
- Call function with jump absolute to 30.
- Return from function with jump absolute.

Reason for failure.

- Need to return to a VARIABLE memory address.

function?

```

10: 8AFF
11: 8BFF
12: C031
13: 1AC0
14: 8BFF
15: C031
16: 9CFF
17: 0000

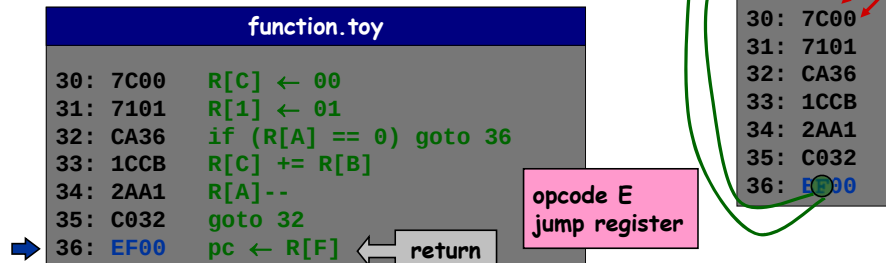
30: 7C00
31: 7101
32: CA36
33: 1CCB
34: 2AA1
35: C032
36: C032
    
```

48

Multiplication Function

Calling convention.

- Jump to line 30.
- Store a and b in registers A and B.
- Return address in register F.
- Put result $c = a \times b$ in register C.
- Register 1 is scratch.
- Overwrites registers A and B.

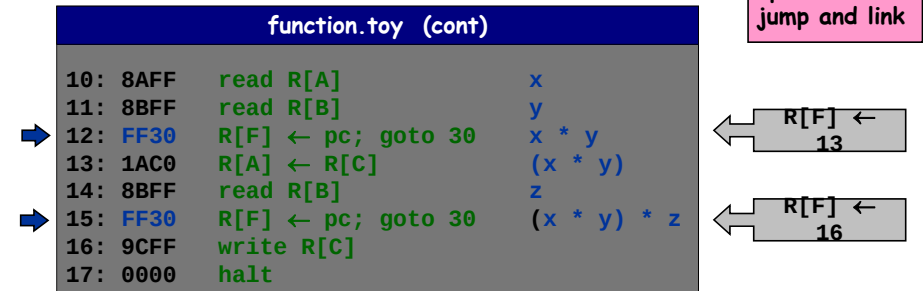


49

Multiplication Function Call

Client program to compute $x \times y \times z$.

- Read x, y, z from standard input.
- Note: PC is incremented before instruction is executed.
- value stored in register F is correct return address



50

Function Call: One Solution

Contract between calling program and function:

- Calling program stores function parameters in specific registers.
- Calling program stores return address in a specific register.
 - jump-and-link
- Calling program sets PC to address of function.
- Function stores return value in specific register.
- Function sets PC to return address when finished.
 - jump register

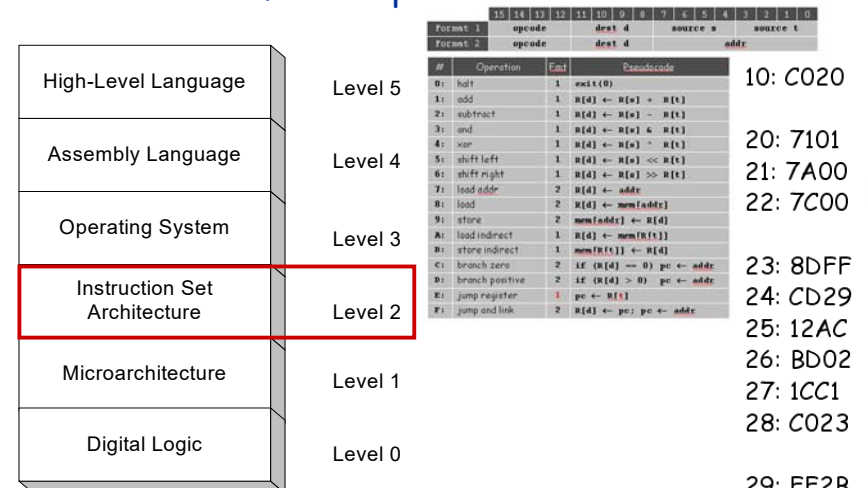
What if you want a function to call another function?

- Use a different register for return address.
- More general: store return addresses on a stack.

51

Virtual machines

Abstractions for computers



52

- Difficult to remember instructions
- Difficult to remember variables
- Hard to calculate addresses/relocate variables or functions
- Need to handle instruction encoding

Table B.1 ARM instruction decode table. (Continued.)

[illegible]

Abstractions for computers

High-Level Language	Level 5	A	DUP	32
Assembly Language	Level 4		<u>lda</u>	R1, 1
			<u>lda</u>	RA, A
			<u>lda</u>	RC, 0
Operating System	Level 3	read	<u>ld</u>	RD, 0xFF
			<u>bz</u>	RD, exit
Instruction Set Architecture	Level 2		<u>add</u>	R2, RA, RC
			<u>sti</u>	RD, R2
Microarchitecture	Level 1		<u>add</u>	RC, RC, R1
			<u>bz</u>	R0, read
Digital Logic	Level 0	exit	<u>jl</u>	RF, <u>println</u>
			<u>hlt</u>	