CISC260 Machine Organization and Assembly Language

CISC 260 Machine Organization and Assembly Language

Spring 2019

Time & Place: 3:30PM-4:45PM, TR, Kirkbride Hall 004

Instructor: Li Liao (Smith 424, 831-3500, liliao@udel.edu), Office Hours: 2:00PM-3:00PM Tuesdays and Thursdays or by appointment

TA: Md Mottalib (Smith ~~203~~ 102A, mmmdip@udel.edu), Office Hours: 12:00PM – 1:00PM, Wednesdays and Thursdays.

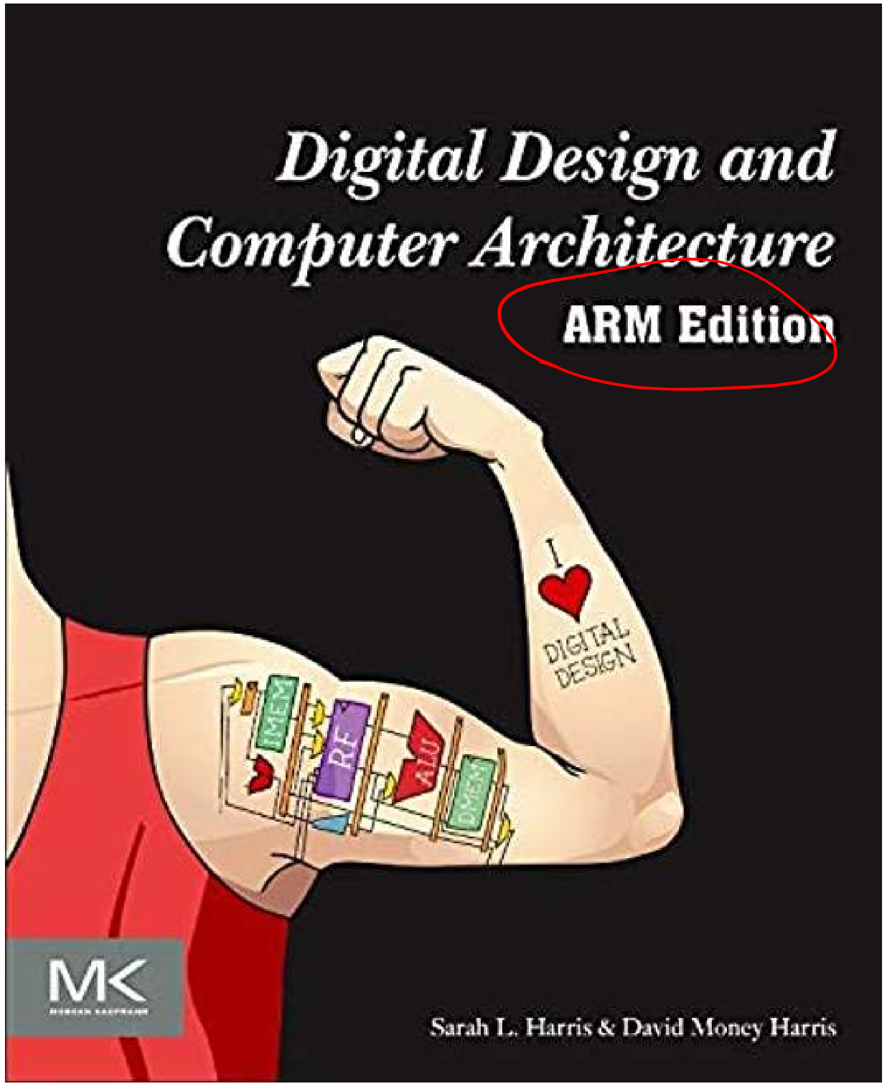
Course Catalog Description:
Introduction to the basics of machine organization. Programming tools and techniques at the machine and assembly levels. Assembly language programming and computer arithmetic techniques.

Assignments and Grading:
There will be 6 homework assignments. All late assignments are subject to 10% penalty per 24 hours past the due time (Saturdays and Sundays do not count), and are not accepted one week past due time. Homework submission is handled electronically via Canvas.

There will be one midterm exam and one final exam. No makeup exams will be granted except when officially acceptable excuses are presented.

All questions about a grade must be presented in writing to the instructor/TA within 1 week since the graded assignment is returned to the class. Afterwards, all grades become final.

Homework 42%, midterm exam 18%, final exam 35%, and participation + quizzes 5%


Digital Design and
Computer Architecture
ARM Edition
I
DIGITAL
DESIGN
IMEM
RF
ALU
DMEM
MK
Sarah L. Harris & David Money Harris

MIPS

- This article **may be too technical for most readers to understand**. Please help improve it to make it understandable to non-experts, without removing the technical details. *(September 2018)*

ARM, previously **Advanced RISC Machine**, originally **Acorn RISC Machine**, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Arm Holdings develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including systems-on-chips (SoC) and systems-on-modules (SoM) that incorporate memory, interfaces, radios, etc. It also designs cores that implement this instruction set and licenses these designs to a number of companies that incorporate those core designs into their own products.

Processors that have a RISC architecture typically require fewer transistors than those with a complex instruction set computing (CISC) architecture (such as the x86 processors found in most personal computers), which improves cost, power consumption, and heat dissipation. These characteristics are desirable for light, portable, battery-powered devices —including smartphones, laptops and tablet computers, and other embedded systems.[3][4][5] For supercomputers, which consume large amounts of electricity, ARM could also be a power-efficient solution.[6]

ARM Holdings periodically releases updates to the architecture. Architecture versions ARMv3 to ARMv7 support 32-bit address space (pre-ARMv3 chips, made before ARM Holdings was formed, as used in the Acorn Archimedes, had 26-bit address space) and 32-bit arithmetic; most architectures have 32-bit fixed-length instructions. The Thumb version supports a variable-length instruction set that provides both 32- and 16-bit instructions for improved code density. Some older cores can also provide hardware execution of Java bytecodes. Released in 2011, the ARMv8-A architecture added support for a 64-bit address space and 64-bit arithmetic with its new 32-bit fixed-length instruction set.[7]

With over 100 billion ARM processors produced as of 2017, ARM is the most widely used instruction set architecture and the instruction set architecture produced in the largest quantity.[8][9][10][11][12] Currently, the widely used Cortex cores, older "classic" cores, and specialized SecurCore cores variants are available for each of these to include or exclude optional capabilities.

Contents [hide]

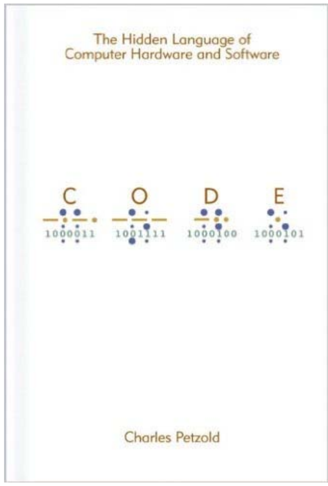
ARM architectures

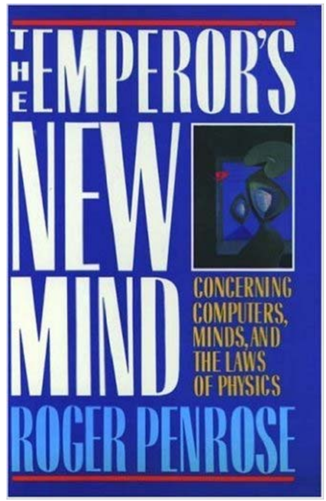
The ARM logo

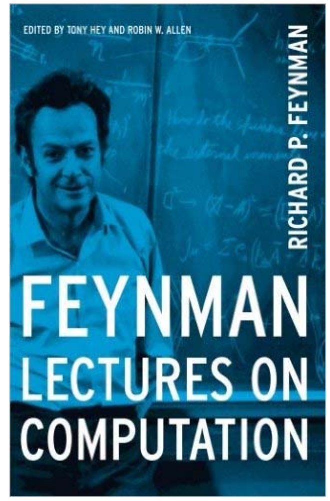
Designer	Arm Holdings
Bits	32-bit, 64-bit
Introduced	1985; 34 years ago
Design	RISC
Type	Register-Register
Branching	Condition code, compare and branch
Open	Proprietary

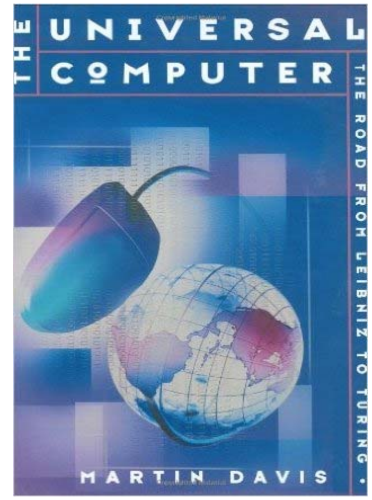
64/32-bit architectures

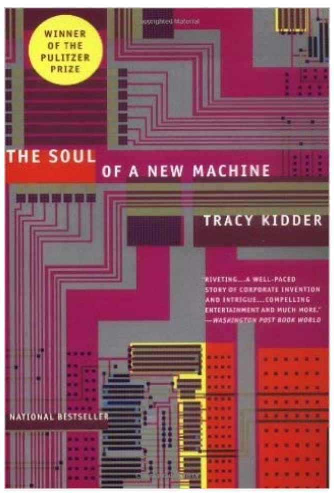
Introduced	2011; 8 years ago
Version	ARMv8-A, ARMv8.1-A, ARMv8.2-A, ARMv8.3-A, ARMv8.4
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]

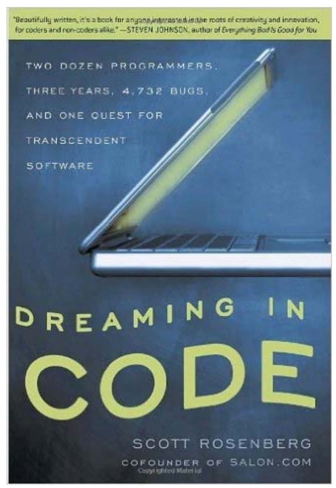

The Hidden Language of
Computer Hardware and Software
C O D E
Charles Petzold

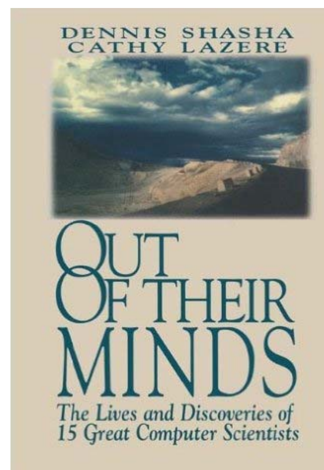

THE EMPEROR'S NEW MIND
CONCERNING COMPUTERS, MINDS, AND THE LAWS OF PHYSICS
ROGER PENROSE

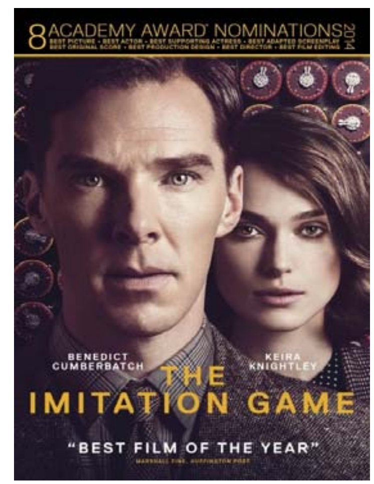

EDITED BY TONY HEY AND ROBIN W. ALLEN
RICHARD P. FEYNMAN
FEYNMAN
LECTURES ON
COMPUTATION


THE UNIVERSAL COMPUTER
THE ROAD FROM LEIBNIZ TO TURING
MARTIN DAVIS


WINNER OF THE PULITZER PRIZE
THE SOUL OF A NEW MACHINE
TRACY KIDDER
NATIONAL BESTSELLER

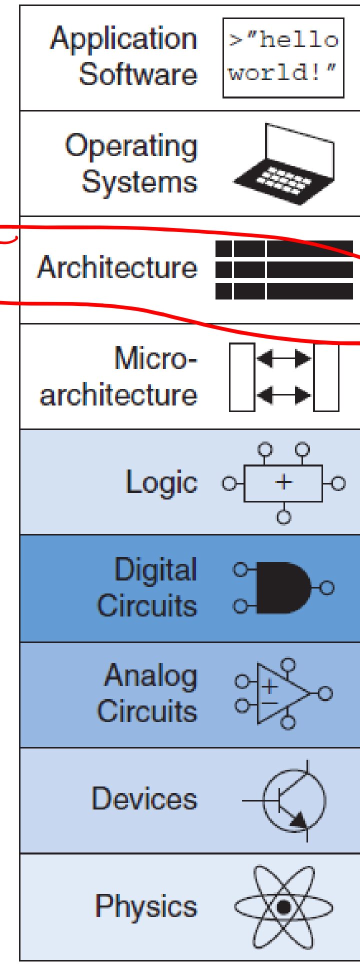

TWO DOZEN PROGRAMMERS,
THREE YEARS, 4,732 BUGS,
AND ONE QUEST FOR
TRANSCENDENT
SOFTWARE
DREAMING IN
CODE
SCOTT ROSENBERG
COFOUNDER OF SALON.COM


DENNIS SHASHA
CATHY LAZERE
OUT OF THEIR MINDS
The Lives and Discoveries of
15 Great Computer Scientists


8 ACADEMY AWARD NOMINATIONS
BENEDICT CUMBERBATCH
KEIRA KNIGHTLEY
THE IMITATION GAME
"BEST FILM OF THE YEAR"

CISC260 Tentative Schedule (S19)

#	Week	Topic
1	Feb11	Overview and Data representation
2	Feb18	Boolean logic, gates
3	Feb25	Build a simple computer
4	Mar4	ARM, ISA, Assembly Language
5	Mar11	Assembly programming
6	Mar18	Procedure call, stack
7	Mar25	Assembly programming and Review1
	Midterm	March 28
8	Apr1	Spring break (no classes)
9	Apr8	Floating point
10	Apr15	Assembly programming: Dynamic data structure
11	Apr22	Assembler, Linker, Compiler
12	Apr29	Performance and optimization
13	May6	I/O and more assembly programming
14	May13	Assembly language programming and security
15	May16	Review2



361

- explain the basic organization of a classical von Neumann machine and its major functional units
- explain how machine code is formatted/organized and executed via the corresponding functional units
- write simple assembly language program segments
- demonstrate how fundamental high-level programming constructs, such as loops, procedure calls and recursions, are implemented at the machine and assembly language level
- convert numerical data between different formats
- carry out basic logical and arithmetic operations
- understand memory management (cache) and basic I/O
- understand performance issues and optimization techniques

C program → Assembly → Machine Code

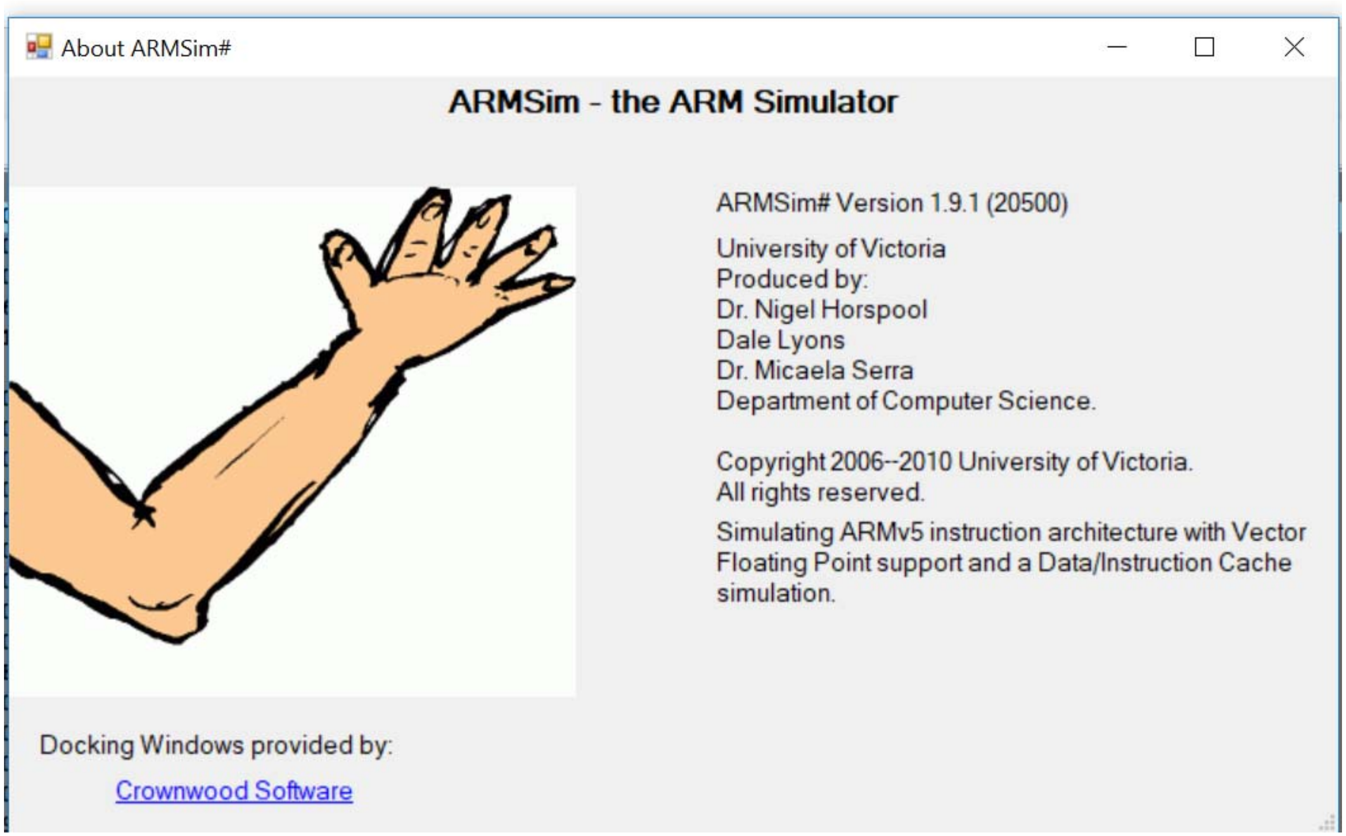
```
gcd(a, b) {
   if(a==b)         return a;
   else if (a>b)    return gcd(a-b, b);
   else             return gcd(a, b-a);
}
```

Compiler

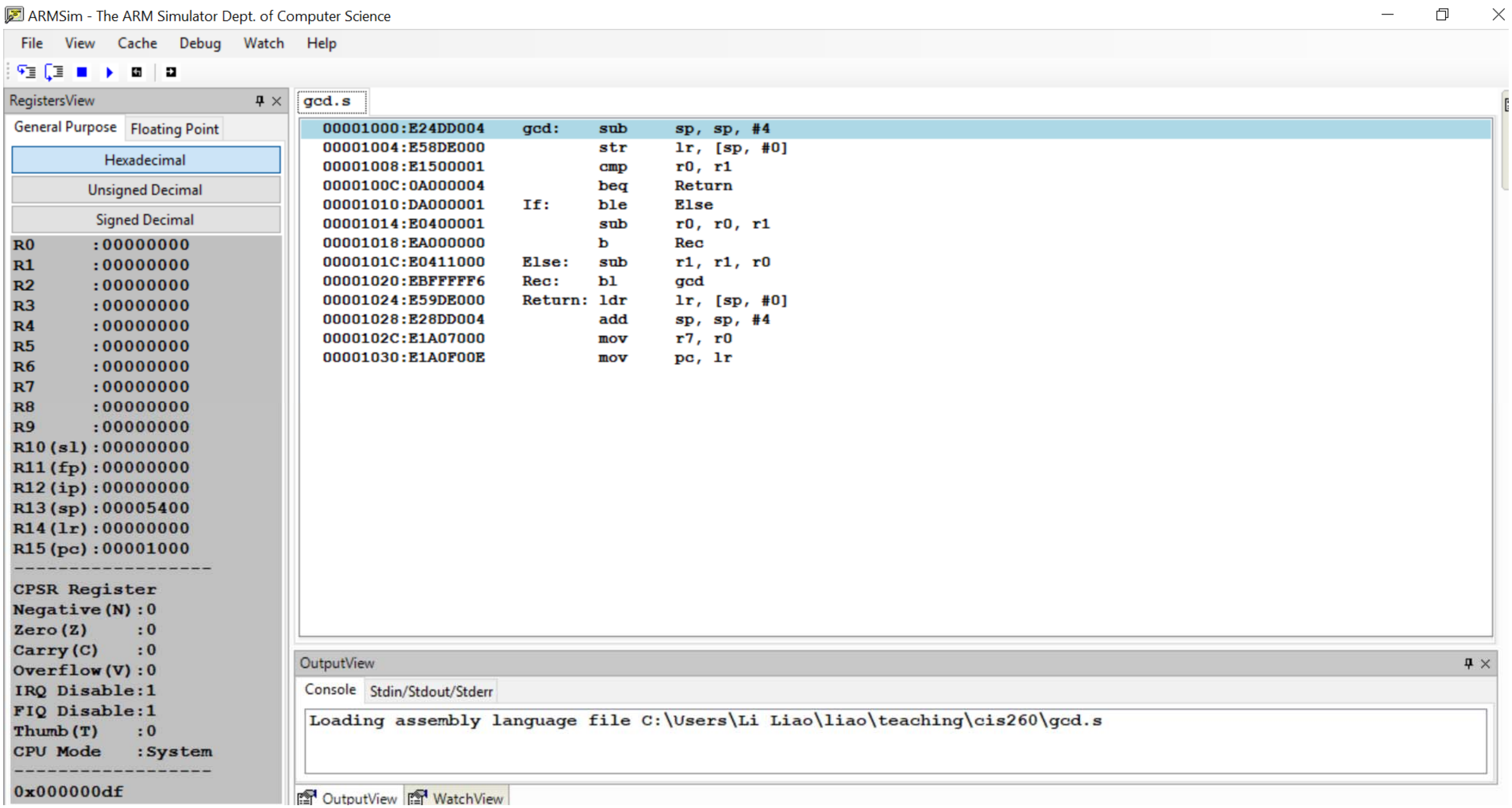
```
gcd:      sub    sp, sp, #4
          str    lr, [sp, #0]
          cmp    r0, r1
          beq    Return
If:       ble    Else
          sub    r0, r0, r1
          b      Rec
Else:     sub    r1, r1, r0
Rec:      bl     gcd
Return:   ldr    lr, [sp, #0]
          add    sp, sp, #4
          mov    r7, r0
          mov    pc, lr
```

Assembler

```
00001000:E24DD004
00001004:E58DE000
00001008:E1500001
0000100C:0A000004
00001010:DA000001
00001014:E0400001
00001018:EA000000
0000101C:E0411000
00001020:EBFFFFF6
00001024:E59DE000
00001028:E28DD004
0000102C:E1A07000
00001030:E1A0F00E
```



https://webhome.cs.uvic.ca/~nigelh/ARMSim-V2.1/index.html


ARMSim - The ARM Simulator Dept. of Computer Science
File View Cache Debug Watch Help
RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00000000
R1 :00000000
R2 :00000000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00005400
R14(lr):00000000
R15(pc):00001000
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T) :0
CPU Mode :System
0x000000df
gcd.s
00001000:E24DD004 gcd: sub sp, sp, #4
00001004:E58DE000 str lr, [sp, #0]
00001008:E1500001 cmp r0, r1
0000100C:0A000004 beq Return
00001010:DA000001 If: ble Else
00001014:E0400001 sub r0, r0, r1
00001018:EA000000 b Rec
0000101C:E0411000 Else: sub r1, r1, r0
00001020:EBFFFFF6 Rec: bl gcd
00001024:E59DE000 Return: ldr lr, [sp, #0]
00001028:E28DD004 add sp, sp, #4
0000102C:E1A07000 mov r7, r0
00001030:E1A0F00E mov pc, lr
OutputView
Console
Stdin/Stdout/Stderr
Loading assembly language file C:\Users\Li Liao\liao\teaching\cis260\gcd.s
OutputView
WatchView

1. Do you have your clicker with you?

A. Yes
B. No

2. How will the following program run?

```
fun() {
        fun();
}
```

A. Run forever
B. Crash by running out of memory
C. It depends on which language is being used
D. Crash in Java
E. Crash in C

3. **What will you get at iteration #31 when running the following program written in C?**

A. 2147483648 (= 2^{31})

B. - 2147483648

C. NaN

D. Crash

E. Don't know

```
#include <stdio.h>

int main() {
int n = 1;
int i = 1;
while (n<50) {
  i = 2 * i;
  printf("iteration%d\t%d\n",n,i);
  n++;
}
return 1;
}
```

4. What will you get at iteration #32 when running the following program written in C?

```
#include <stdio.h>

int main() {
int n = 1;
int i = 1;
while (n<50) {
  i = 2 * i;
     printf("iteration%d\t%d\n",n,i);
  n++;
}
return 1;
}
```

A. 2^{32}

B. -2^{32}

C. –2147483648

D. 0

D. Crash

5. What will be printed by the following java code?

```
double x1 = 0.3;
double x2 = 0.1 + 0.1 + 0.1;
StdOut.println(x1 == x2);
```

A. True
B. False