CS1021 Introduction to Computing I
0. Introduction

Rebekah Clarke
clarker7@scss.tcd.ie

Motivation

"The performance of future software systems will be dramatically affected ... by how well software designers understand the basic hardware techniques at work in a system"

David A. Patterson and John L. Hennessy

"A person who is more than casually interested in computers should be well schooled in machine language, since it is a fundamental part of a computer."

Donald E. Knuth

Objectives

On successful completion of CS1021 you will be able to:

- Describe the basic characteristics, structure and operation of a computer system;
- Represent and interpret basic information (integers, text) in binary form;
- Translate between simple high-level programming language constructs and their assembly language equivalents;
- Design, construct, document and test small-scale assembly language programs to solve simple problems;
- Reason about the cost of executing instructions and the efficiency of simple programs;
- Make use of appropriate documentation and reference material.

Assembly Language

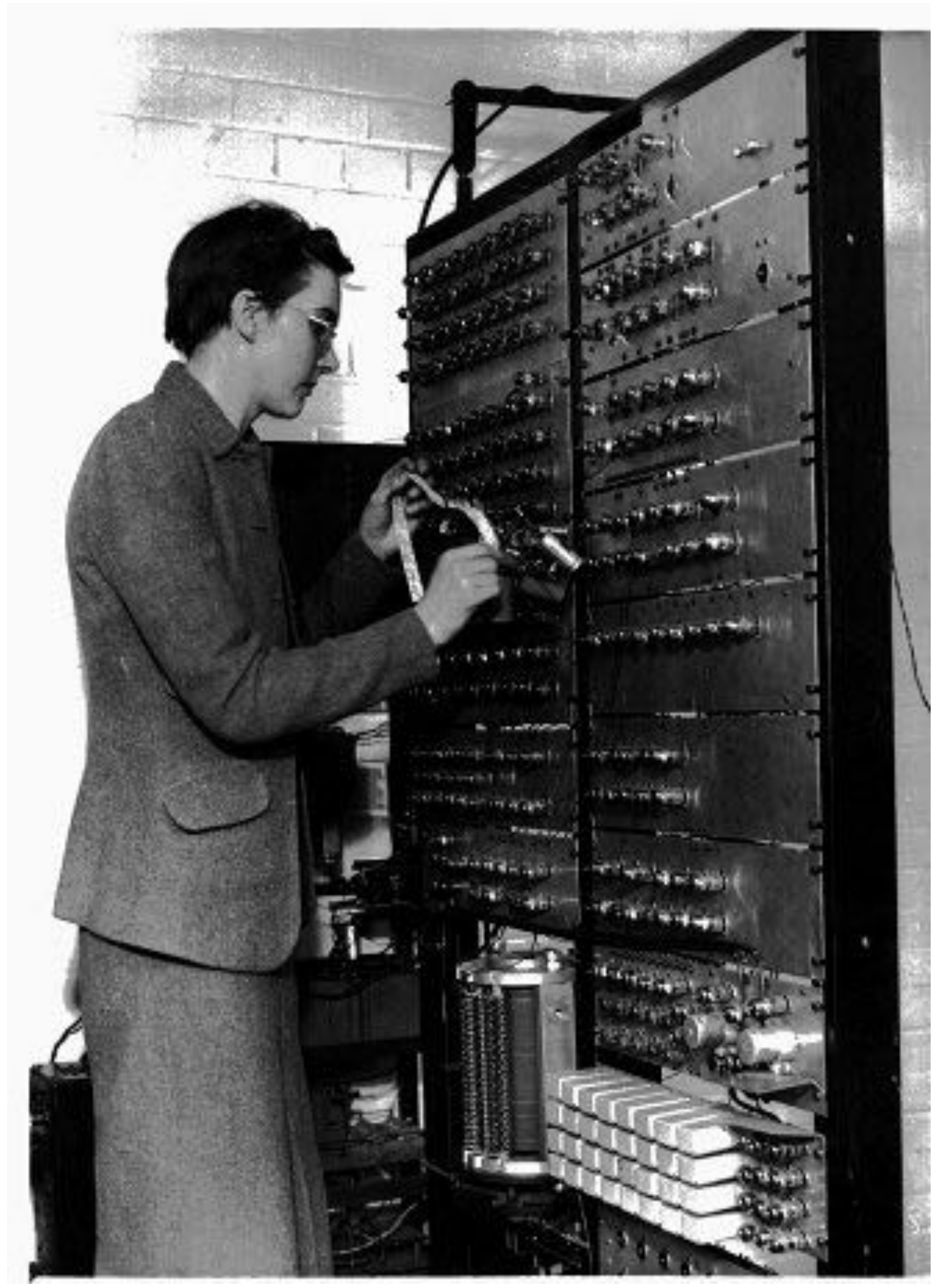
First developed in 1947 by Kathleen Booth

Removed the need for programers to memorise lengthly codes or make manual address calculations

Once used to write entire operating systems

Now has niche uses particularly in optimising for program speed or size

Used for device drivers, flight navigation systems, extremely high security situations, mobile firmware…



Timetable

Lectures

Timetable subject to change

Monday 14:00 in MacNeil

Tuesday 11:00 in LB01

Tutorials

Thursday 9:00 Goldsmith or 13:00 in LB04

Check which tutorial you should attend on Blackboard (available by Thursday)

No tutorial in Week 1

Labs

Friday 10:00, 11:00, 12:00, 13:00 in LG35/LG36 (O'Reilly Institute)

There <u>will</u> be labs in Week 1

Check which lab you should attend on Blackboard (available Thursday)

Assessment

CS1021
Introduction to Computing I

- ICS, CSB & CSL
- 5 credits
  (out of 60 for the full year of your degree course)
- Labs (4)
- Assignments (2)
- 2 hour examination
  (worth 70% of CS1021)

CS1022
Introduction to Computing II

- ICS only
- 5 credits
  (out of 60 for the full year of your degree course)
- Labs (4)
- Assignments (1-2)
- 2 hour examination
  (worth 70% of CS1022)

Attendance and Deadlines

Attendance at all lectures, labs and tutorials is compulsory

- Attendance will be strictly taken at all tutorials and labs
- You MUST attend your own lab session

Catch up as quickly as you can (by working through lecture notes, tutorials and lab exercises in your own time)

Zero marks for late coursework without explanation

Email the course TA, Aimee Borda (bordaa@tcd.ie), if you will miss a lab or tutorial

Inform your tutor if you will miss a major deadline or will be absent for more than a day

You may be returned as Non-Satisfactory (see College Calendar) if you miss more than one-third of your Lectures, Labs or Tutorials, if you fail to submit more than one third of your Lab Exercises or if you fail to submit either term Assignment.

Laptops etc.

Laptops, tablets etc. may not be used during lectures

There are exceptions (e.g. if you are registered with the College Disability Service and require the use of a laptop, just let me know)

You may use a laptop / tablet during tutorials for referring to lecture material, documentation, etc. only

You may use your laptop / tablet during labs

Recipe for Success

Do all of the coursework (tutorials / labs / assignments)

Do the coursework <u>yourself</u>

If you don’t understand something, spend a little time studying it

If necessary, get help from classmates demonstrators or lecturer

Don’t wait before seeking help

Don’t wait for someone to tell you that …

- you haven’t handed in coursework
- you don’t understand something
- you haven’t been attending lectures / tutorials / labs

Don’t think that if you don’t understanding something now, you can fix it later before the exam … <u>this module doesn’t work that way!</u>

Your Own Work

When you submit work as part of an exercise or assignment, you are implying that it is **your own work**

DO indicate where you received help from someone other than a lecturer, teaching assistant or demonstrator

DO indicate where you have used other sources of information (e.g. websites or text books)

DON'T share your work with other students – in your year or any other year – you will make it harder for them to succeed in College

DO discuss your work with each other, ask another student for hints to solve problems, ask for assistance fixing bugs, etc.

DO be prepared to explain any work that you submit and expect us to use plagiarism detection tools such as TurnItIn

Taking credit for someone else's work without giving them due recognition is a serious academic offence (**plagiarism**, see your course handbook)

Simple Model of a Microprocessor System

A **processing unit** or **processor** which performs operations on data

Memory, which stores:

Data: representing text, images, videos, sensor readings, π, audio, etc. ...

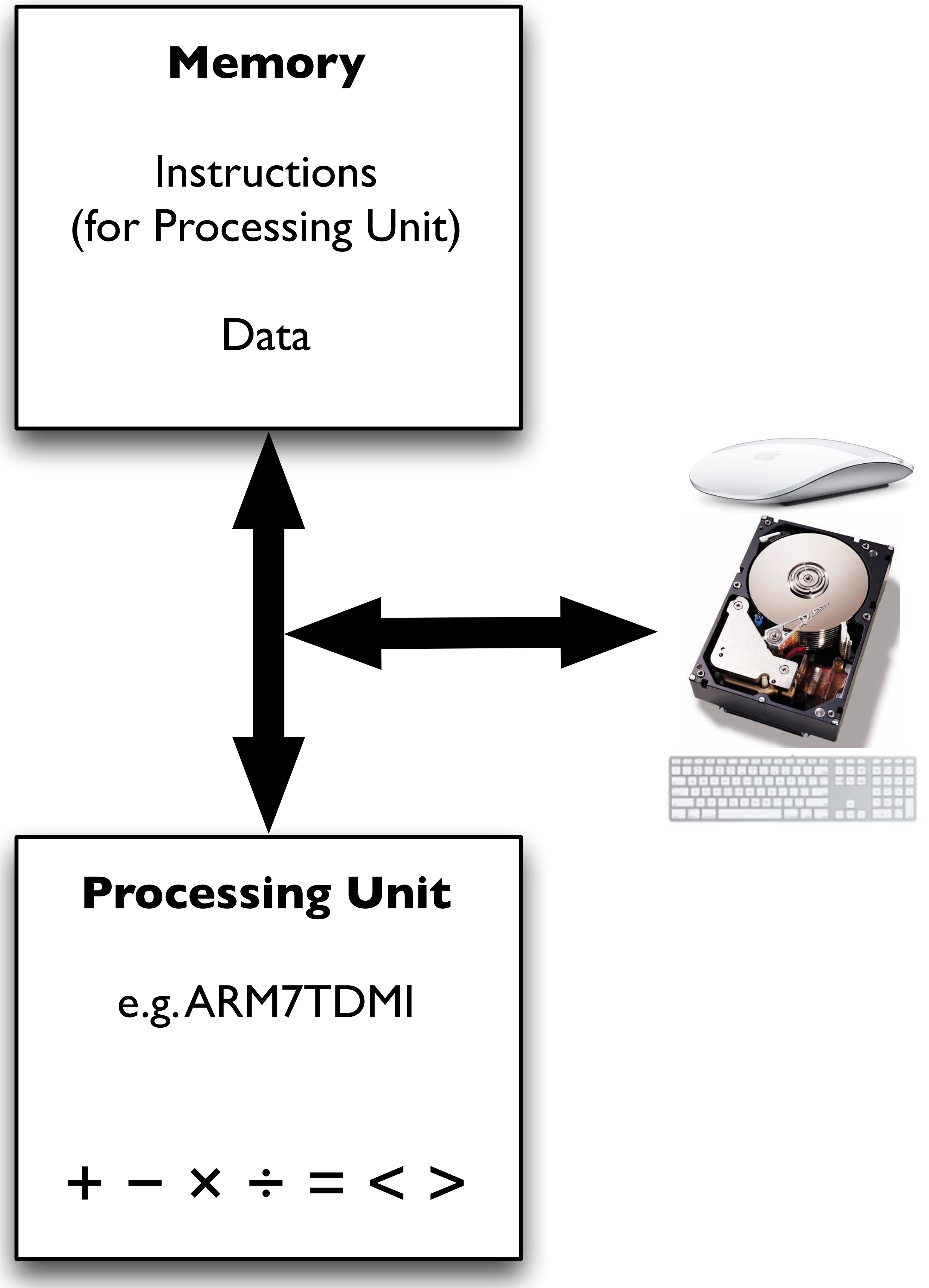
Instructions: Programs are composed of sequences of instructions that control the actions of the processing unit

Instructions typically describe very simple operations, e.g.

Add two values together

Move a value from one place to another

Compare two values



Simple Model of a Microprocessor System

Memory is arranged as a series of "locations"

Each location has a unique "**address**"

- e.g. the memory location at address 21000 contains the value 78

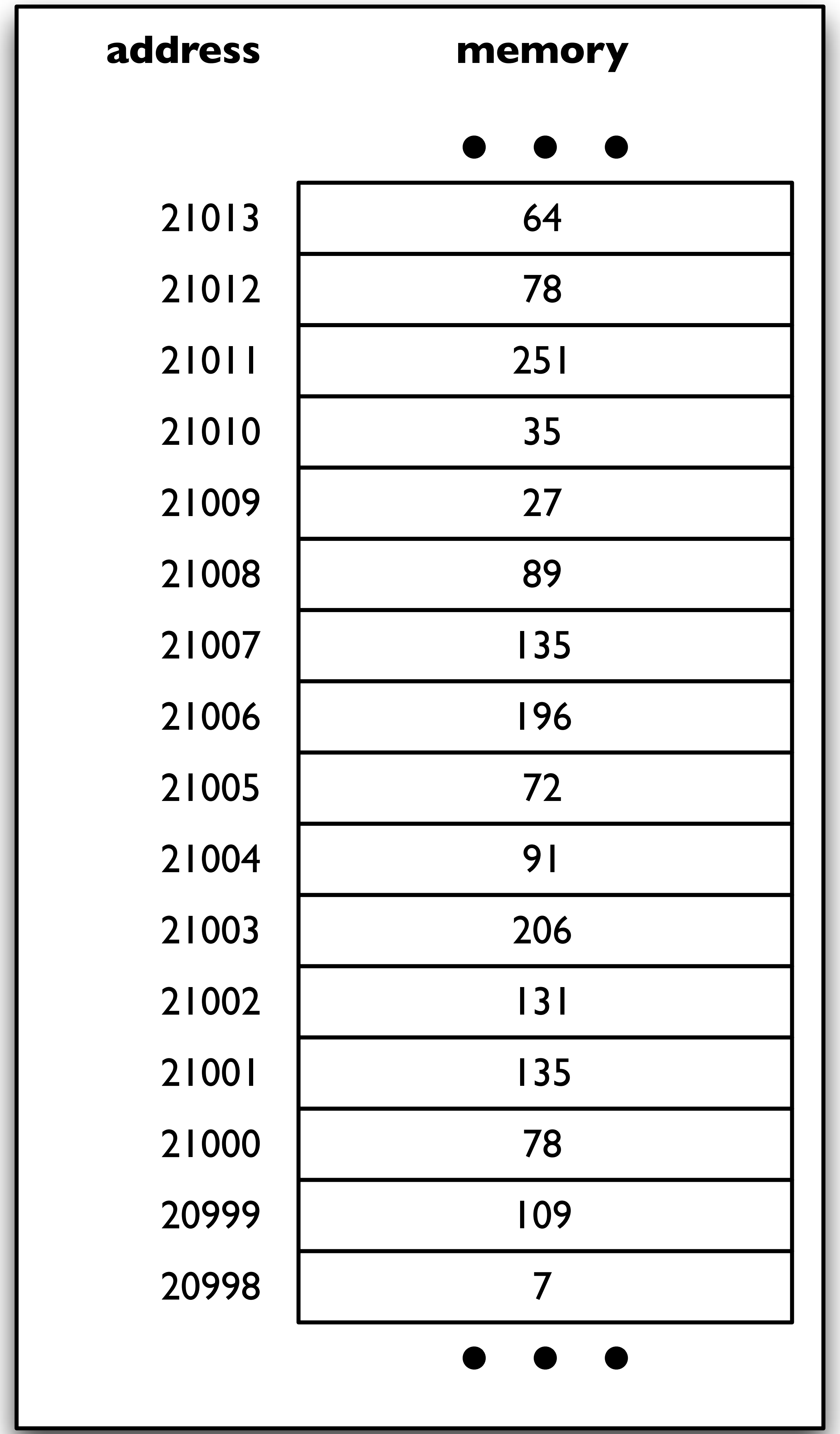
The number of locations in memory is limited

- e.g. 2GB of RAM 2,147,483,648 locations!

Each location can contain either data or an instruction

Instructions are encoded as values

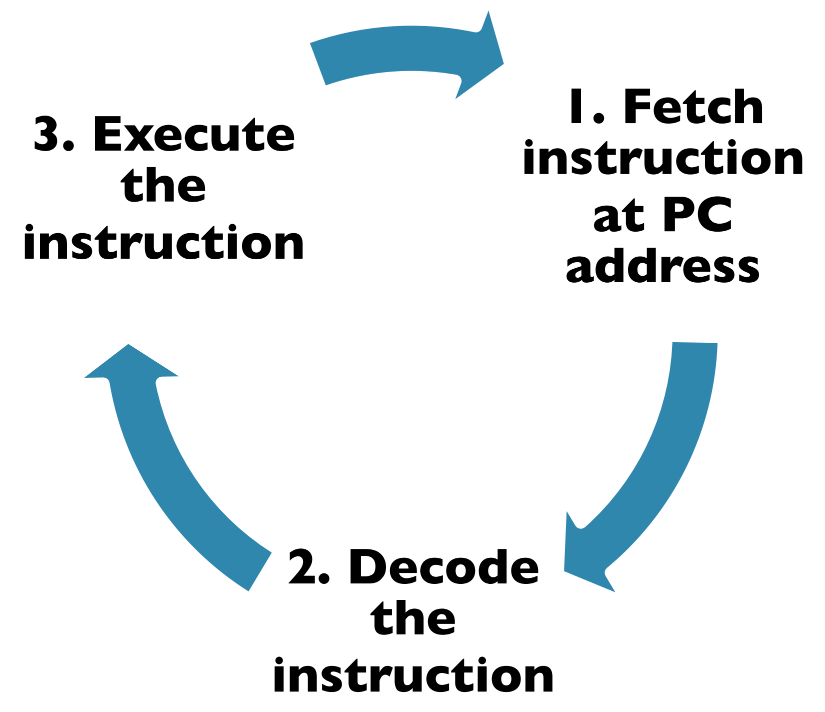
- e.g. the value 91 might be the code used to tell the processing unit to add two values together

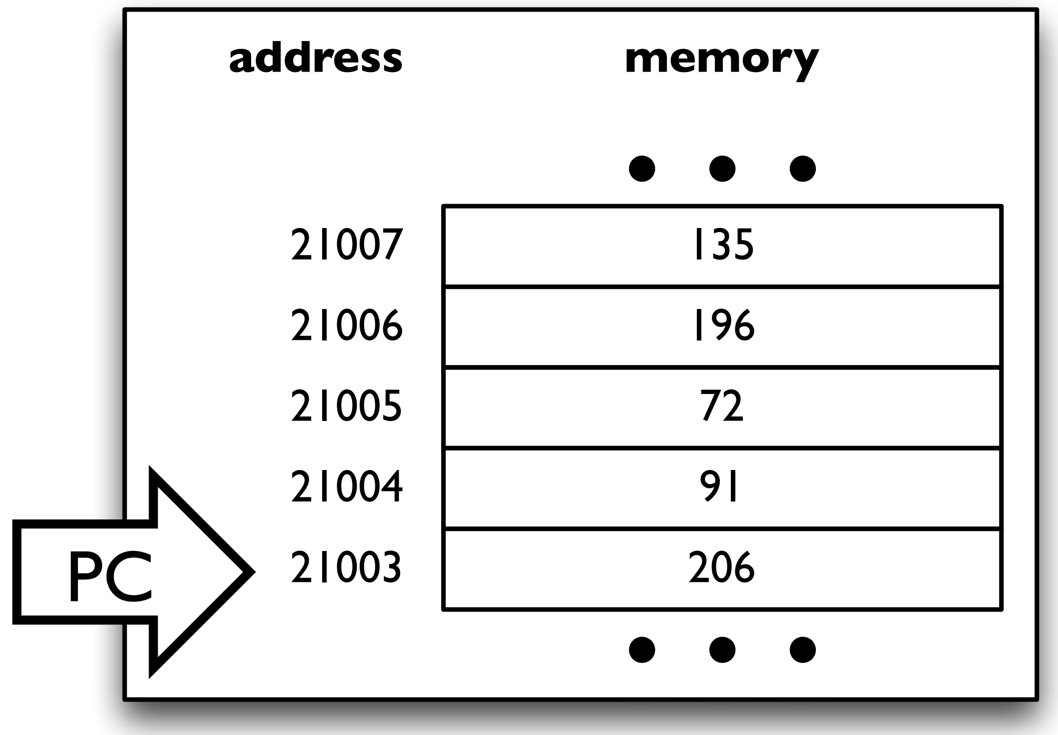


address	memory
	• • •
21013	64
21012	78
21011	251
21010	35
21009	27
21008	89
21007	135
21006	196
21005	72
21004	91
21003	206
21002	131
21001	135
21000	78
20999	109
20998	7
	• • •

Program Execution

When the computer is turned on, the processing unit begins executing the instruction in memory at the address stored in the Program Counter or PC


1. Fetch instruction at PC address
2. Decode the instruction
3. Execute the instruction



After fetching an instruction, the value of the Program Counter is changed to the address of the next instruction in the program

Processing unit keeps doing this until the computer is turned off

Program Execution

This simple model of a programmable computer is the model used by computers familiar to us (PCs, games consoles, mobile phones, engine management units, ...)

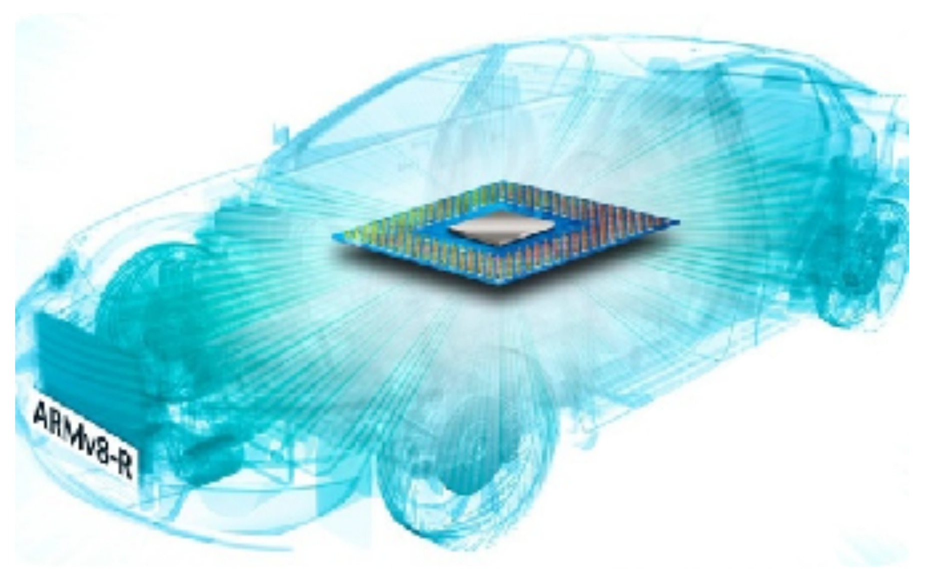
Behaviour is entirely predictable

The "power" of computers arises because they perform a lot of simple operations very quickly

The complexity of computers arises because useful programs are composed of many thousands or millions of simple instructions

- Possibly executing in parallel on more than one processor/computer!

ARM Architecture

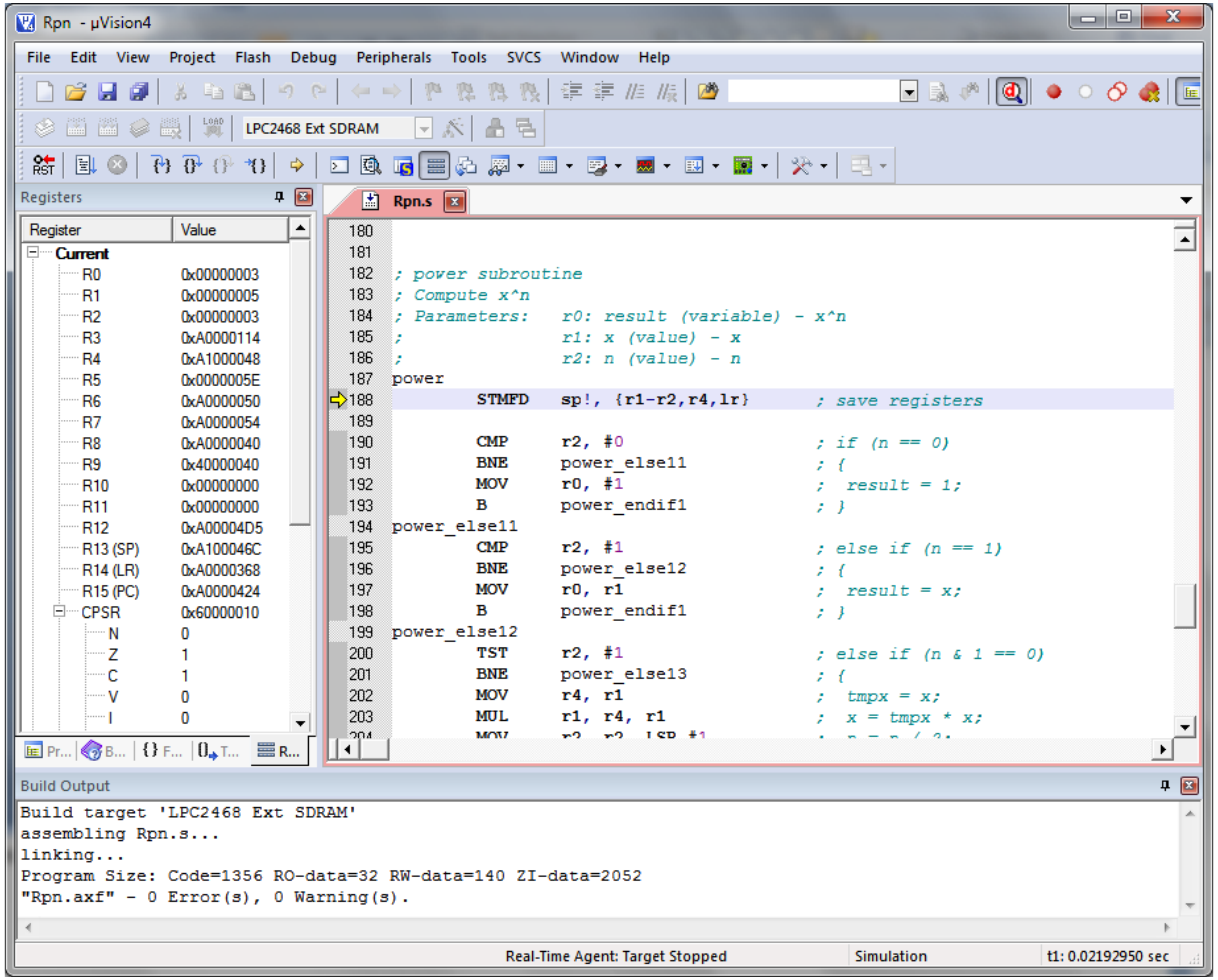


Origins of ARM

“Steve is one of the brightest guys I've ever worked with – brilliant and when we decided to do a microprocessor on our own I made two great decisions - I gave them two things which National, Intel and Motorola had never given their design teams: the first was no money; the second was no people. The only way they could do it was to keep it really simple.”

Hermann Hauser, founder of Acorn Computers
(ARM Holdings is a former subsidiary of the now defunct Acorn Computers)

Development Environment



Register	Value
Current	
R0	0x00000003
R1	0x00000005
R2	0x00000003
R3	0xA0000114
R4	0xA1000048
R5	0x0000005E
R6	0xA0000050
R7	0xA0000054
R8	0xA0000040
R9	0x40000040
R10	0x00000000
R11	0x00000000
R12	0xA00004D5
R13 (SP)	0xA100046C
R14 (LR)	0xA0000368
R15 (PC)	0xA0000424
CPSR	0x60000010
N	0
Z	1
C	1
V	0
I	0

```


; power subroutine
; Compute x^n
; Parameters:   r0: result (variable) - x^n
;               r1: x (value) - x
;               r2: n (value) - n
power
        STMFD   sp!, {r1-r2,r4,lr}              ; save registers

        CMP     r2, #0                          ; if (n == 0)
        BNE     power_else11                    ; {
        MOV     r0, #1                          ;  result = 1;
        B       power_endif1                    ; }
power_else11
        CMP     r2, #1                          ; else if (n == 1)
        BNE     power_else12                    ; {
        MOV     r0, r1                          ;  result = x;
        B       power_endif1                    ; }
power_else12
        TST     r2, #1                          ; else if (n & 1 == 0)
        BNE     power_else13                    ; {
        MOV     r4, r1                          ;  tmpx = x;
        MUL     r1, r4, r1                      ;  x = tmpx * x;
        MOV     r2, r2, LSR #1                  ;  n = n / 2;
```

```
Build target 'LPC2468 Ext SDRAM'
assembling Rpn.s...
linking...
Program Size: Code=1356 RO-data=32 RW-data=140 ZI-data=2052
"Rpn.axf" - 0 Error(s), 0 Warning(s).
```

Real-Time Agent: Target Stopped   Simulation   t1: 0.02192950 sec

Demonstration

- Keil µVision Development Environment
- Write a simple program
- “Assemble” the program
- Load the program into memory and debug it
- Observe the results

Demonstration

A simple program that adds four numbers:

1. Make the first number our total
2. Add the second number to the total
3. Add the third number to the total
4. Add the fourth number to the total

Demonstration

```
start
        MOV   total, a               ; Make the first number the subtotal
        ADD   total, total, b        ; Add the second number to the subtotal
        ADD   total, total, c        ; Add the third number to the subtotal
        ADD   total, total, d        ; Add the fourth number to the subtotal
```

Call the numbers R1, R2, R3, R4

Call the total R0

Demonstration

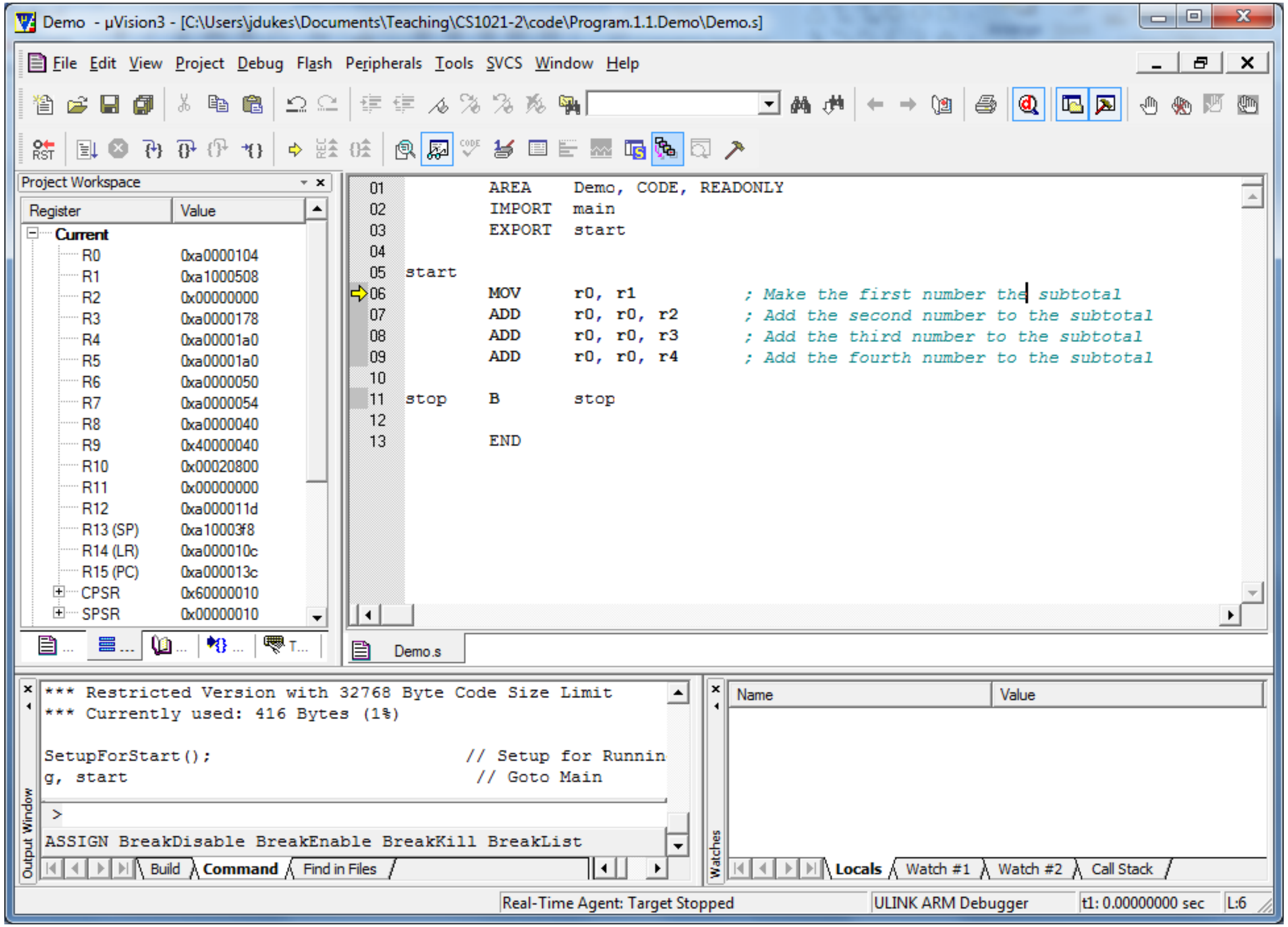
```
        AREA    Demo, CODE, READONLY
        IMPORT  main
        EXPORT  start

start
        MOV   R0, R1               ; Make the first number the subtotal
        ADD   R0, R0, R2           ; Add the second number to the subtotal
        ADD   R0, R0, R3           ; Add the third number to the subtotal
        ADD   R0, R0, R4           ; Add the fourth number to the subtotal

stop    B     stop

        END
```

Development Environment


Demo - µVision3 - [C:\Users\jdukes\Documents\Teaching\CS1021-2\code\Program.1.1.Demo\Demo.s]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
Demo.s
Output Window
Build
Command
Find in Files
Watches
Locals
Watch #1
Watch #2
Call Stack
Real-Time Agent: Target Stopped
ULINK ARM Debugger
t1: 0.00000000 sec
L:6

Register	Value
Current	
R0	0xa0000104
R1	0xa1000508
R2	0x00000000
R3	0xa0000178
R4	0xa00001a0
R5	0xa00001a0
R6	0xa0000050
R7	0xa0000054
R8	0xa0000040
R9	0x40000040
R10	0x00020800
R11	0x00000000
R12	0xa000011d
R13 (SP)	0xa10003f8
R14 (LR)	0xa000010c
R15 (PC)	0xa000013c
CPSR	0x60000010
SPSR	0x00000010

```
        AREA    Demo, CODE, READONLY
        IMPORT  main
        EXPORT  start

start

        MOV     r0, r1              ; Make the first number the subtotal
        ADD     r0, r0, r2          ; Add the second number to the subtotal
        ADD     r0, r0, r3          ; Add the third number to the subtotal
        ADD     r0, r0, r4          ; Add the fourth number to the subtotal

stop    B       stop

        END
```

```
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 416 Bytes (1%)

SetupForStart();                          // Setup for Runnin
g, start                                   // Goto Main

>
ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Name	Value