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

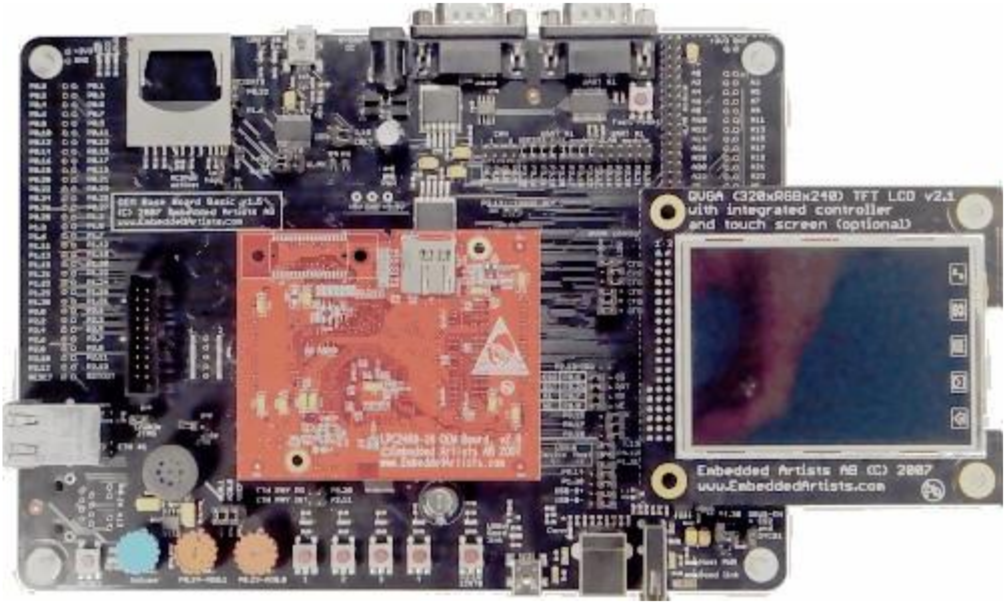
Objectives

- On successful completion of 3D1 you will be able to:
  - describe the basic characteristics, structure and operation of a microprocessor system;
  - 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.

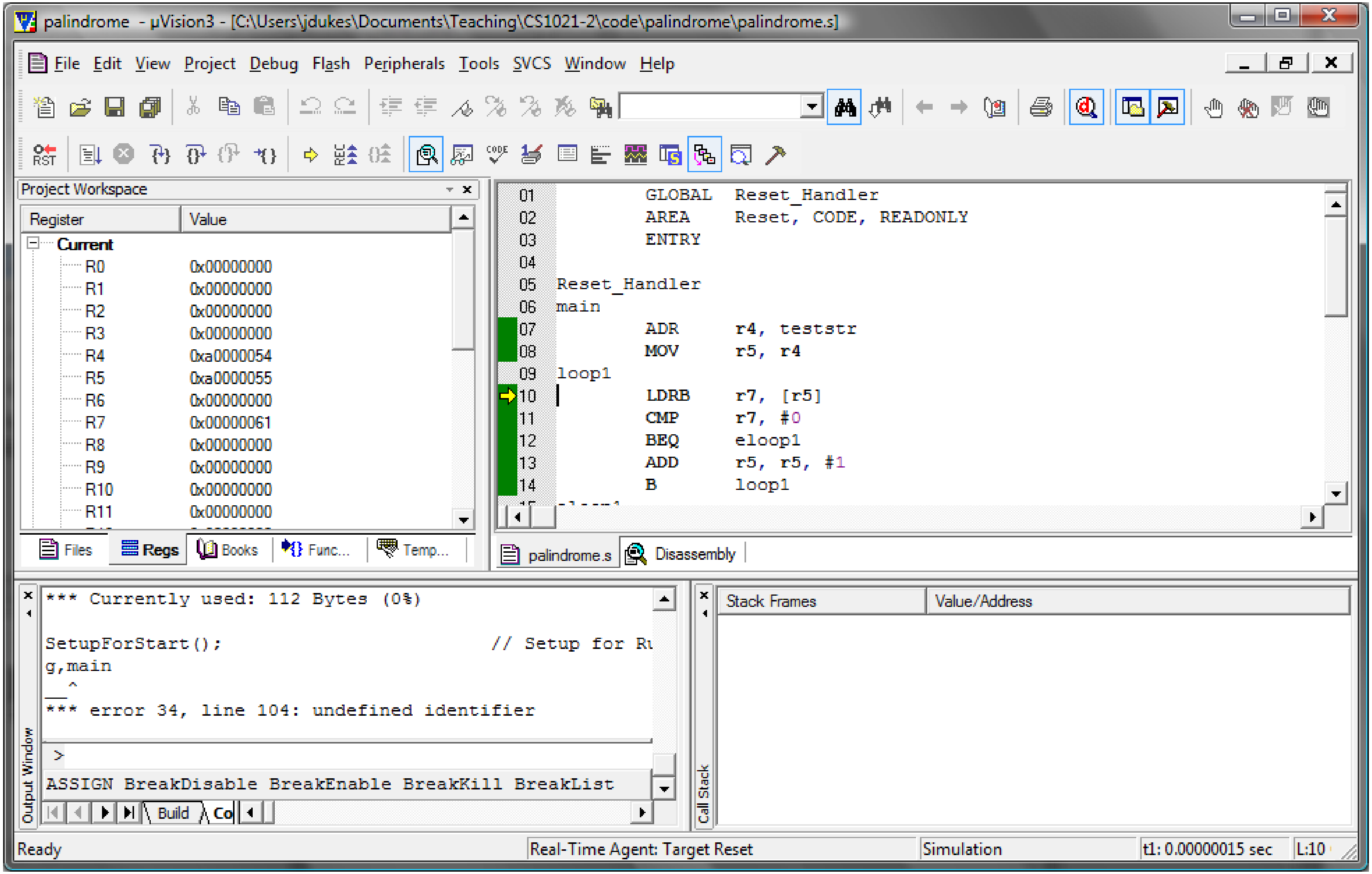
ARM

ARM7TDMI

- iPod, Nintendo DS, Nokia mobiles, Lego Mindstorms, ...
- NXP LPC2468 32-bit microcontroller
  - ARM7TDMI-S CPU
  - Flash memory (512KiB), RAM (96KiB)
  - 10/100 Ethernet, USB 2.0, A/D & D/A converters, ...


QVGA (320xRGBx240) TFT LCD v2.1
with integrated controller
and touch screen (optional)
Embedded Artists AB (C) 2007
www.EmbeddedArtists.com

Development Environment



palindrome - µVision3 - [C:\Users\jdukes\Documents\Teaching\CS1021-2\code\palindrome\palindrome.s]

Register	Value
Current	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0xa0000054
R5	0xa0000055
R6	0x00000000
R7	0x00000061
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000

```
        GLOBAL  Reset_Handler
        AREA    Reset, CODE, READONLY
        ENTRY

Reset_Handler
main
        ADR     r4, teststr
        MOV     r5, r4
loop1
        LDRB    r7, [r5]
        CMP     r7, #0
        BEQ     eloop1
        ADD     r5, r5, #1
        B       loop1
```

```
*** Currently used: 112 Bytes (0%)

SetupForStart();                        // Setup for Ru
g,main
__^
*** error 34, line 104: undefined identifier

>
ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Stack Frames	Value/Address

Ready | Real-Time Agent: Target Reset | Simulation | t1: 0.00000015 sec | L:10

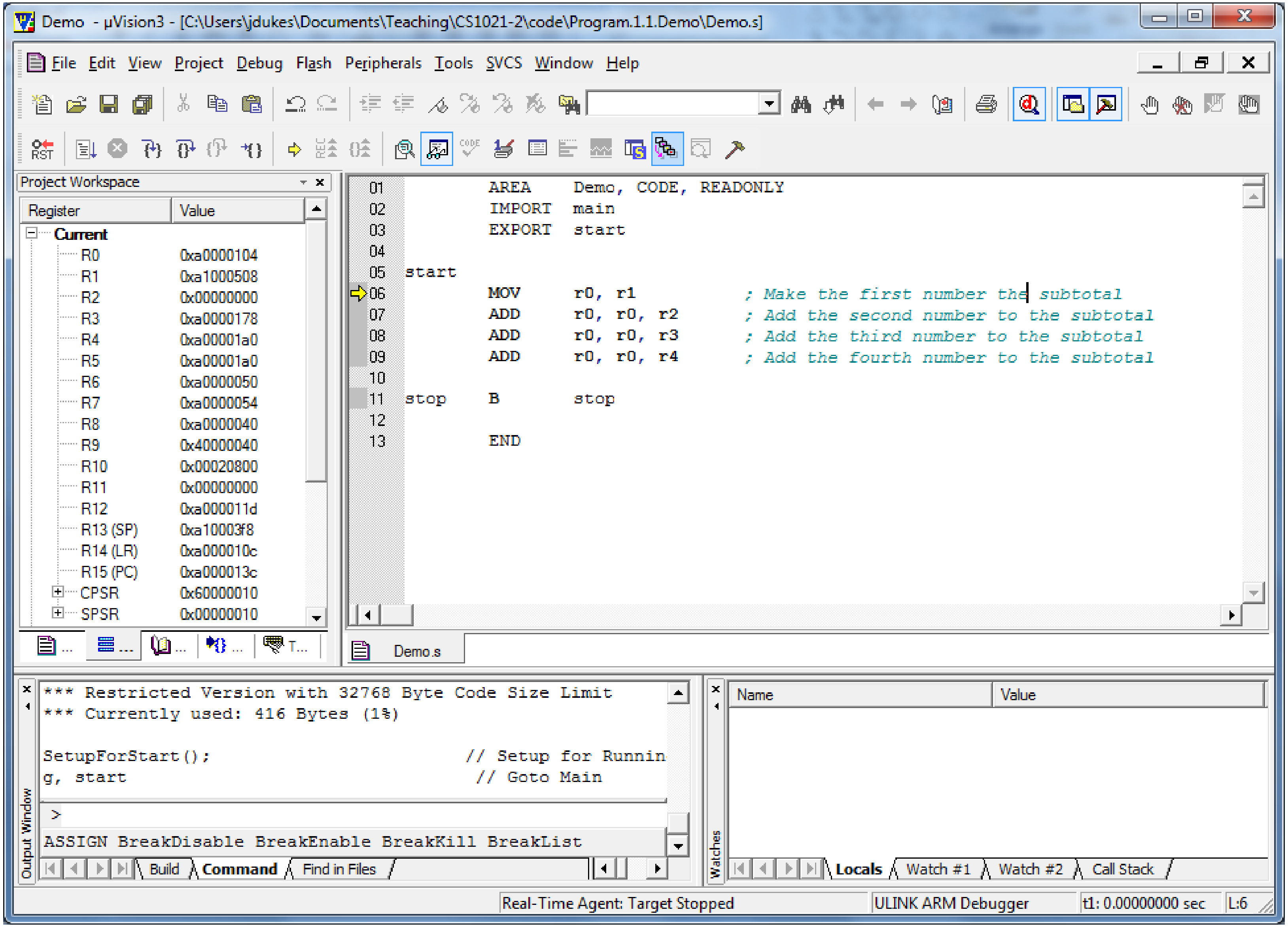
Demonstration

- Keil µVision Development Environment
- Writing a simple program
- “Building” the program
- Loading the program into memory and debugging it
- Observing the results

Demonstration

- A simple program that adds four numbers
  - Make the first number our subtotal
  - Add the second number to the subtotal
  - Add the third number to the subtotal
  - Add the fourth number to the subtotal

Demonstration


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
RST
Project Workspace
Demo.s
Output Window
Build
Command
Find in Files
Watches
Locals
Watch #1
Watch #2
Call Stack

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

Real-Time Agent: Target Stopped | ULINK ARM Debugger | t1: 0.00000000 sec | L:6

Program 1.1 – 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
```

Reading – ARM Assembly Language

- Recommended reading
  - William Hohl, *"ARM Assembly Language: Fundamentals and Techniques"*, CRC Press, 2009.

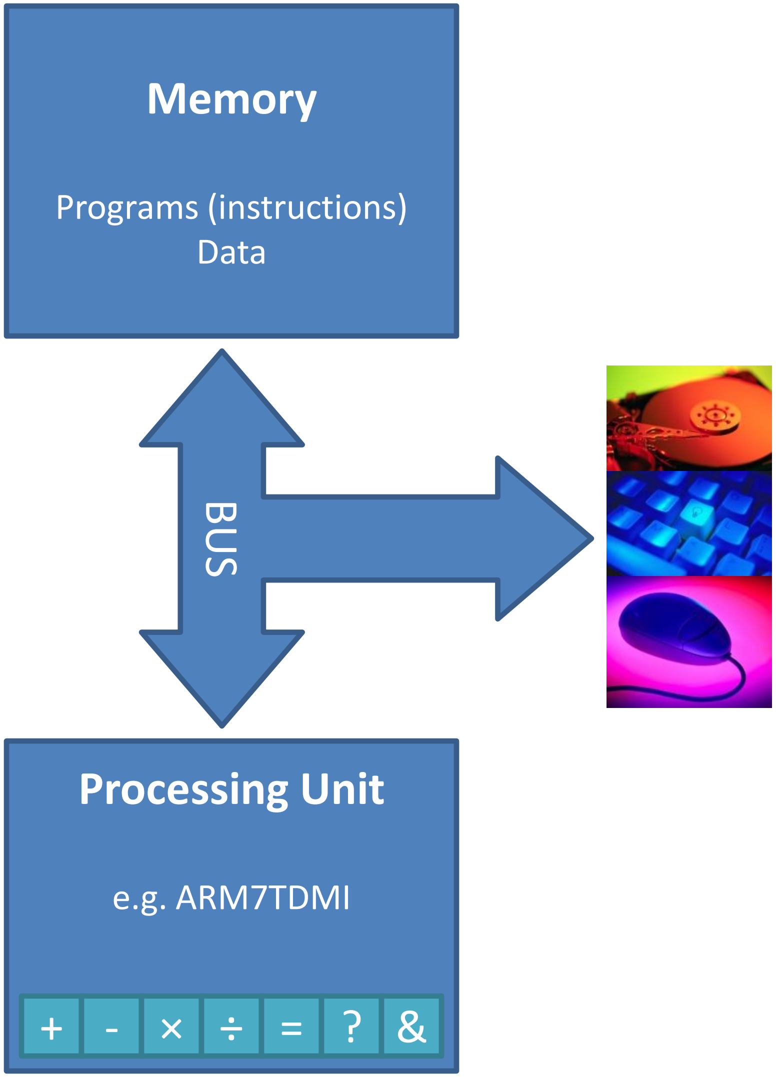
- Other reading
  - Andrew Sloss, Dominic Symes and Chris Wright, *"ARM System Developer's Guide: Designing and Optimizing System Software"*, Morgan Kaufmann, 2004.
  - Steve Furber, *"ARM System-on-Chip Architecture"*, 2nd edition, Addison-Wesley Professional, 2000.
  - Peter Knaggs, Stephen Welsh, *ARM: Assembly Language Programming*, Bournemouth University, 2004

Reading – Computer Architecture

- Other reading
  - David A. Patterson and John L. Hennessy, *“Computer Organization and Design: The Hardware / Software Interface”*, 4th edition, Morgan Kaufmann, 2009. *(introductory text)*
  - John L. Hennessy and David A. Patterson, *“Computer Architecture: A Quantitative Approach”*, 4th edition, Morgan Kaufmann, 2007. *(advanced text – for later years)*

Simple Model of a Microprocessor System

- A **Processing Unit** 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



Memory

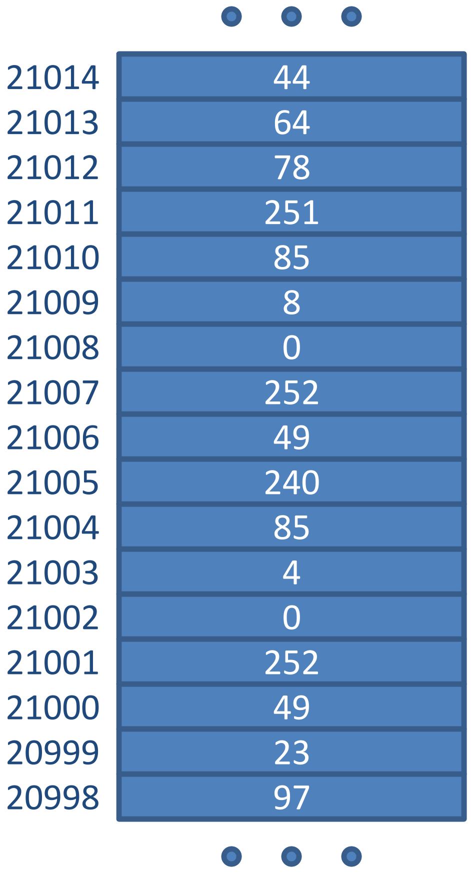
Programs (instructions)
Data

Processing Unit

e.g. ARM7TDMI

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 **49**
- 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 49 might be the code used to tell the processor to add two values together



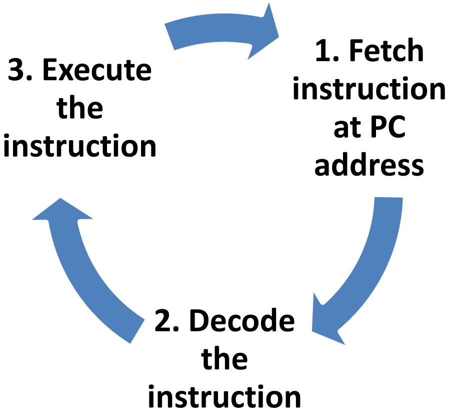
• • •

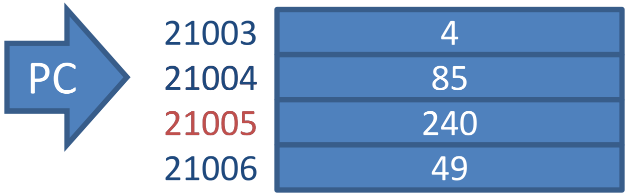
21014	44
21013	64
21012	78
21011	251
21010	85
21009	8
21008	0
21007	252
21006	49
21005	240
21004	85
21003	4
21002	0
21001	252
21000	49
20999	23
20998	97

• • •

Simple Model of a Microprocessor System

- 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



21003	4
21004	85
21005	240
21006	49

  - After executing an instruction, the value of the Program Counter is changed to the address of the next instruction in the program
  - The processing unit keeps doing this until the computer is turned off

Simple Model of a Microprocessor System

- 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 (**deterministic**)
  - *If that's the case, how can computers generate random numbers?*
- 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 computer!*