DESN2000: Engineering Design & Professional Practice (EE&T)

Week 1

Introduction to embedded systems, ARM architecture and assembly language fundamentals

David Tsai

School of Electrical Engineering & Telecommunications

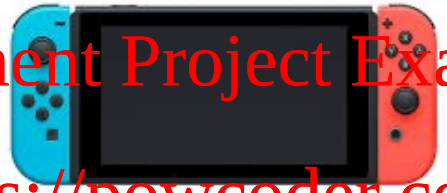
Graduate School of Biomedical Engineering

d.tsai@unsw.edu.au

Biomedical Microsystems Lab

Why embedded systems?

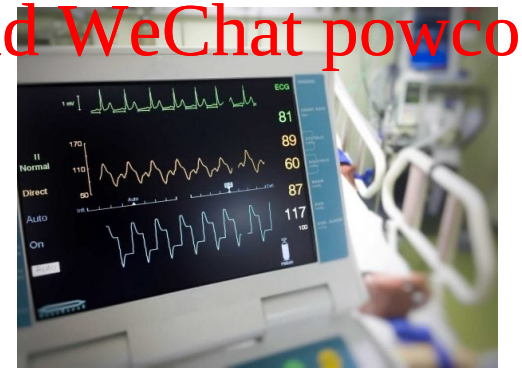
iPhone



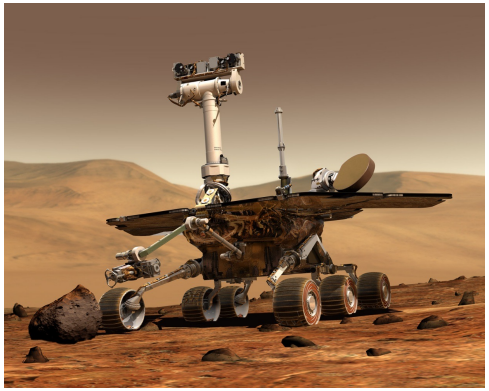
Nintendo Switch

Network router

Oscilloscope



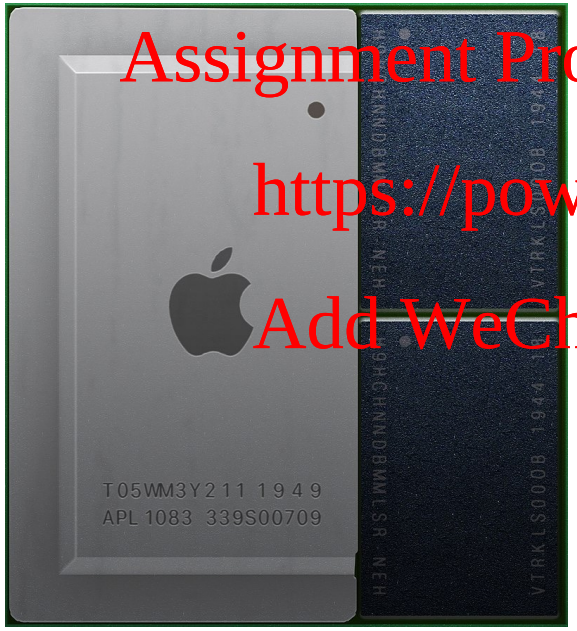
Patient monitors



Mars Pathfinder

Why ARM?

- Dominates the embedded device market (> 95% of smart phones) .
- Increasingly used in PCs (the new Macs) and servers (Amazon).



Apple M1 ARMv8.6-A (Nov 2020)

Apple A14 ARMv8.5-A (Oct 2020)

Technical objectives

- Understanding embedded systems, using ARM architecture as example
- Programming embedded systems
- Interfacing hardware with embedded microprocessors

Course staff

Course coordinator (Design Next)
Ilpo Koskinen ilpo.koskinen@unsw.edu.au

School coordinator (EE&T)
David Tsai d.tsai@unsw.edu.au

Lecturers (Design Next)
Arianna Vignati a.vignati@unsw.edu.au
Nicholas Gilmore nicholas.gilmore@unsw.edu.au

Head lab demonstrator (EE&T)
Jonah Meggs j.meggs@unsw.edu.au

Lab demonstrators (EE&T)
Rebekah Rae z5206258@ad.unsw.edu.au
Jack Murray jack.murray@unsw.edu.au
Shai Alaloff z5311773@ad.unsw.edu.au
Sabine Seeto z5312137@ad.unsw.edu.au
Samuel Chapman z5218557@ad.unsw.edu.au
Shiqi (Charlotte) Han z5312977@ad.unsw.edu.au
Ronakraj Gosalia r.gosalia@unsw.edu.au
Zhuoyu (Tony) Chen z5210863@ad.unsw.edu.au
Bradley Lin z5258949@ad.unsw.edu.au
Riley Dean z5308666@ad.unsw.edu.au
Jonathan Podiono z5278938@ad.unsw.edu.au
Zachary Milgate z5015456@ad.unsw.edu.au
Sumanth Devathi z5160042@ad.unsw.edu.au
Oishik Sarkar o.sarkar@unsw.edu.au

Lectures

- EE&T: weekly 3-hour lectures, using Blackboard Collaborate (via Moodle).
- Design Next: on some weeks only
- Conservation Biologist Guest Lecture: Week 3, Tuesday 14th June, 1 – 2 PM.
  - A biologist's perspective on bird tracking (your design project).
  - Simon Gorta
    Centre for Ecosystem Science
    School of Biological, Earth and Environmental Sciences (BEES)
    UNSW
  - Lecture is recorded, OK to follow-up after the live lecture. But has Q&A, so attend if you can.

EE&T weekly exercises

- Not assessed.
- Answers provided the following week.
- Will announce a weekly consultation to go through some of these (likely Tuesday 12:30-13:00). Will be recorded in Blackboard Collaborate.
- Takes 1 ~ 1.5 hr. DO THEM! FINAL EXAM QUESTIONS ARE SIMILAR TO THESE.

Week	Contents
1	Number systems
2	Number systems and ARM architecture
3	Data processing and memory access
4	Control flow
5	Functions
6	
7	I/O interfacing
8	I/O interfacing
9	Pseudo-instructions and directives
10	

EE&T laboratory

- 3 hours at ElecEng119 (local students) or online via Teams (overseas students).
- Working in pairs.
- Starts week 3, or **week 2 for those in Monday labs** (Monday Week-3 is public holiday).
- For remote students: remote access using Teams, webcam, internet-enabled oscilloscopes. Demonstrators will communicate with you using Teams.

Week	Contents
1	
2	1 – Introduction to the QVGA base board, µVision and debugging (for students in Mon. labs)
3	1 – Introduction to the QVGA base board, µVision and debugging
4	2.1 – Data types, control flow, assembly programming
5	2.2 – Data types, control flow, assembly programming
6	3 – Functions and subroutines
7	4 – I/O
8	5.1 – D/A conversion
9	5.2 – D/A conversion
10	6 – LCD & touchscreen

Assessments

Assessment	Contribution
EE&T laboratory exercises	20%
Final exam	30%
Project	
Design journal	25%
Design presentation	15%
Code implementation	10%

- EE&T laboratory attendance and exercises are **compulsory**.
  - If you missed your labs, schedule catch-up with Jonah Meggs (j.meggs@unsw.edu.au), with medical certificate.
  - Competing less than 7 of 8 labs = automatic DESN2000 failure, regardless of course mark. No exception.

Getting help

- Fastest: ask your demonstrators during the lab / workshop.
- Moodle Forum actively monitored by lecturers and demonstrators.

- Lab / workshop logistics: Jonah Meggs (head lab demo; j.meggs@unsw.edu.au)
- Design Next lectures and assessments: TBA
- Elec. lectures and assessments: Dr David Tsai (d.tsai@unsw.edu.au)
  Don't use MS Teams… get buried among the dozens of alerts I get each day.

This week

- Introduction to embedded systems
- ARM7TDMI programmer's model
- ARM instructions and tools
- Assembly language examples
- Assembler directives and operators

Embedded systems: introduction

- Traditionally: a **microprocessor-based system**, having **limited resources** and dedicated to a **specific task**.
- Computers are also microprocessor-based systems but are general purpose devices.

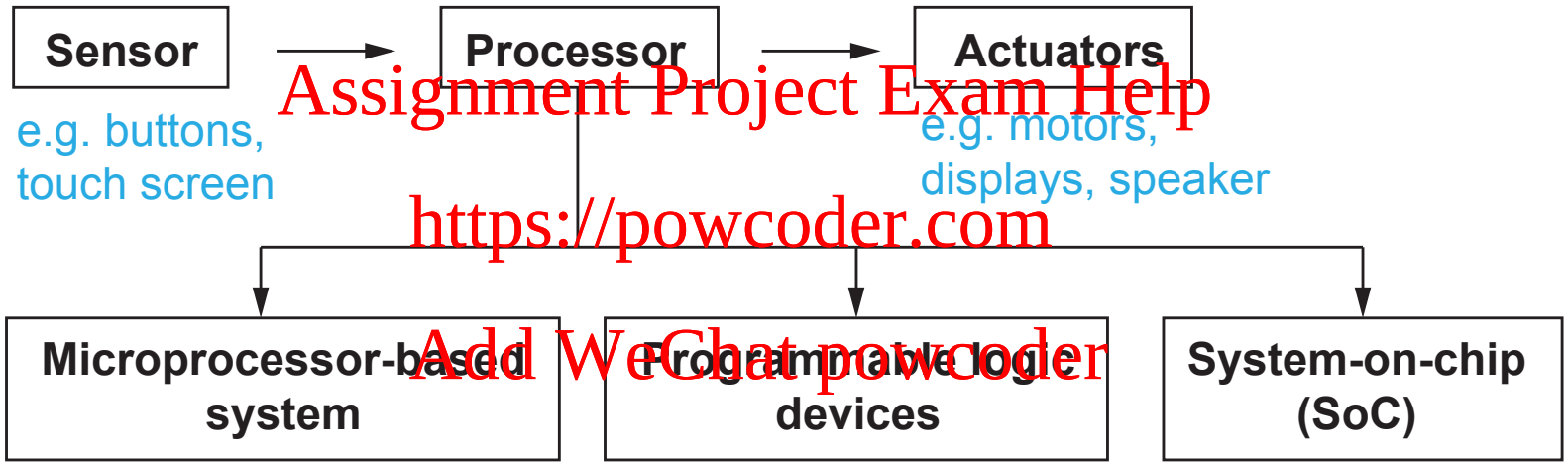
- Embedded systems are typically:
  1. Single purpose
  2. Cost- and resource-sensitive
  3. Real-time constraints
  4. Limited by power, speed and area constraints
  5. Small code size (a washing machine controller with 16 KB)

Embedded systems: types

e.g. buttons, touch screen

e.g. motors, displays, speaker

Embedded systems: types

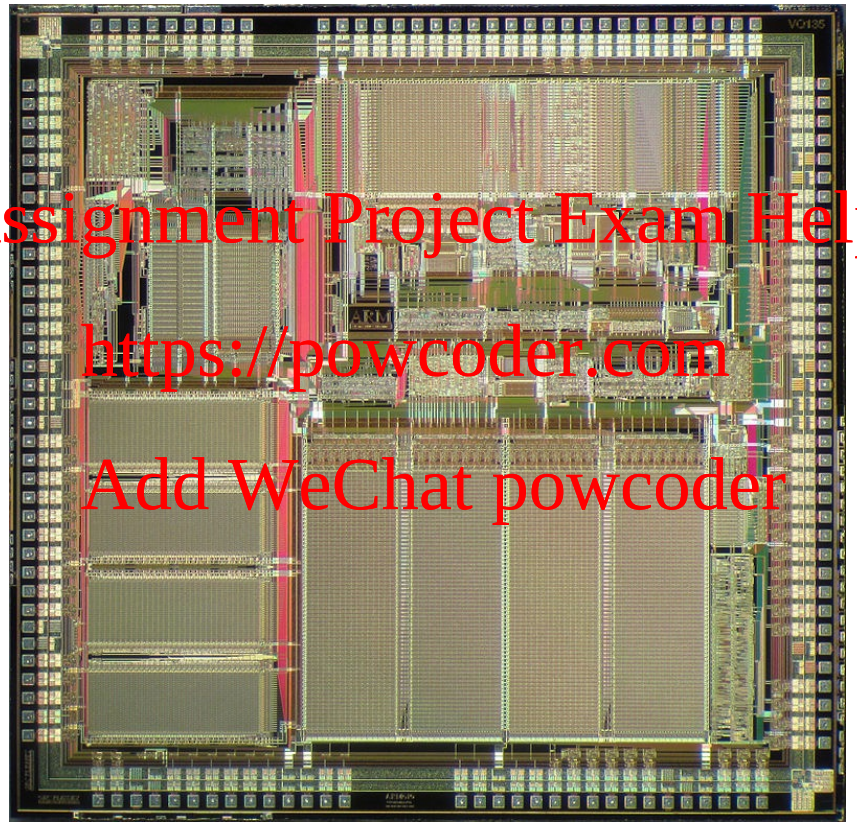


Embedded systems: microprocessors

- The brain of an embedded system is the microprocessor.
- A microprocessor has:
  - Arithmetic Logic Unit (ALU)
  - Registers and internal bus structure
  - Control unit (CU) – can be hardwired or microprogrammed

Embedded systems: microprocessors

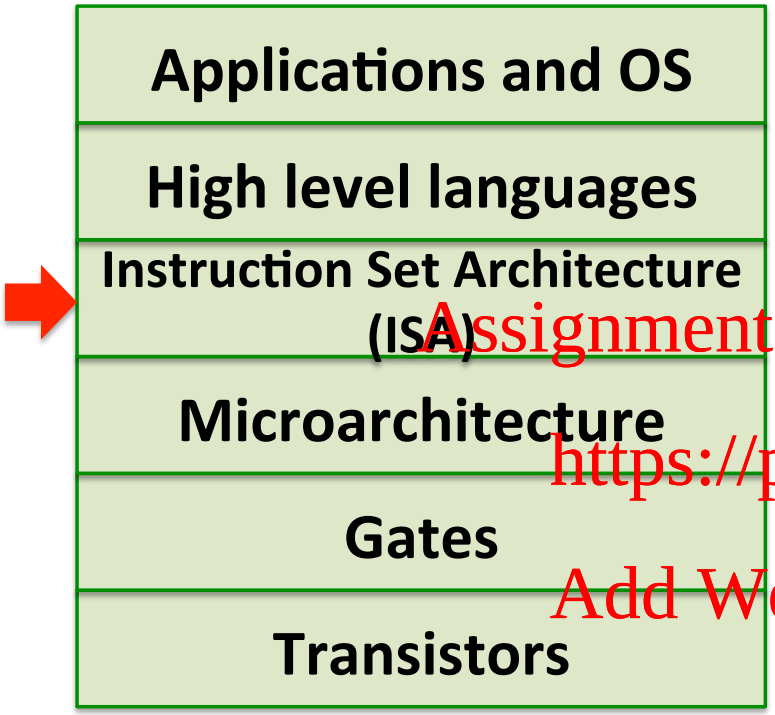
- Computation occurs at the transistor level: electrons running around metal traces and semiconductors.



ARM610 processor die photo

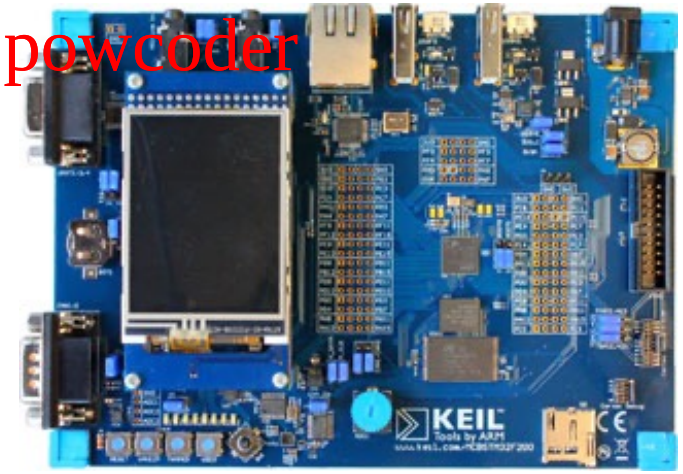
- Convenient to have a high-level abstraction.

ARM7TDMI programmer's model



In this course:

- ARM7TDMI processor architecture
  - Implements ARMv4 instruction set
- LPC2478 chip on QVGA board
- Keil MDK-ARM uVision4 IDE
- Assembly language programming



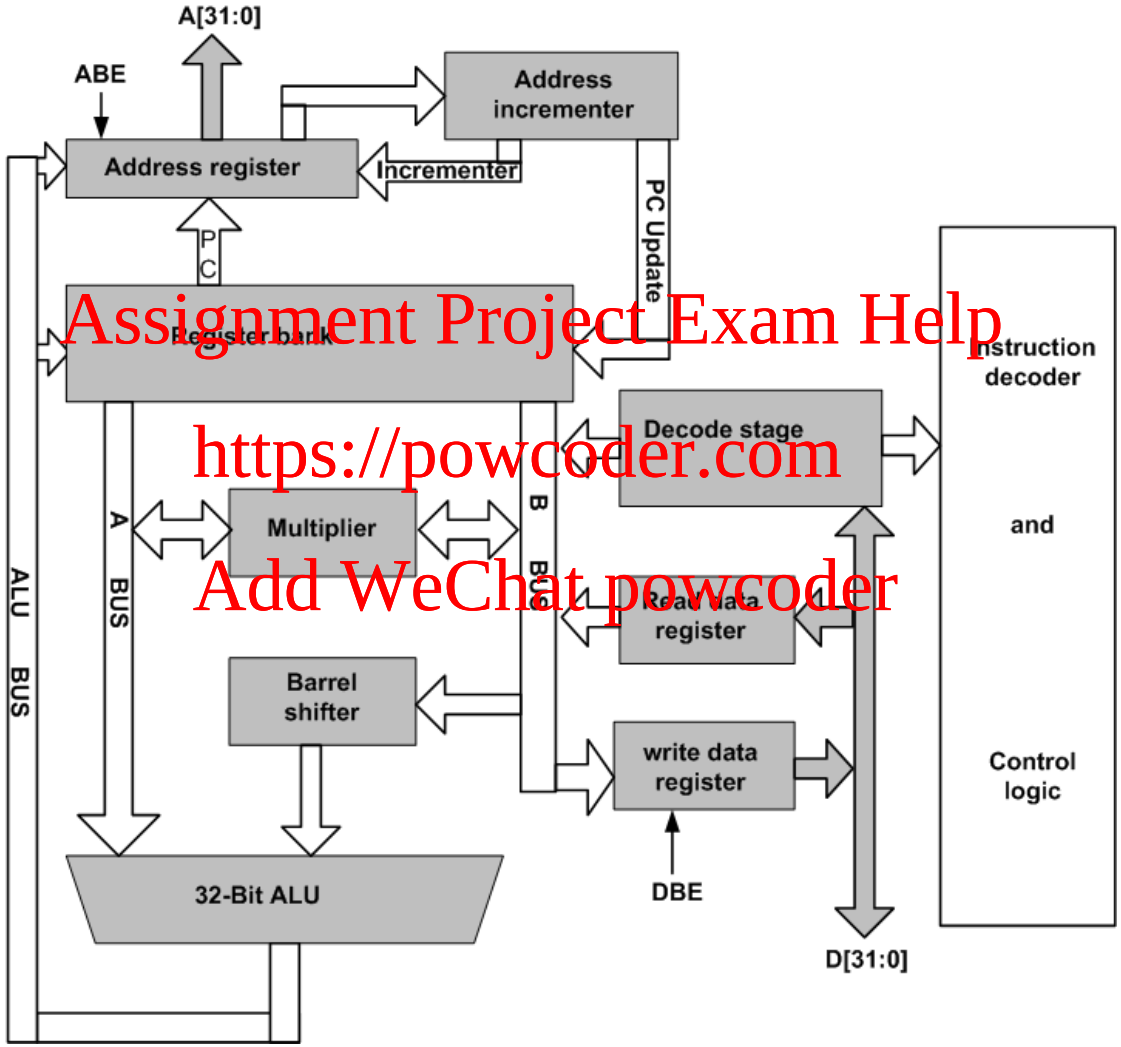
ARM7TDMI: T: thumb, D: debug, M: multiplier, I: in-circuit emulation

ARM7TDMI programmer's model

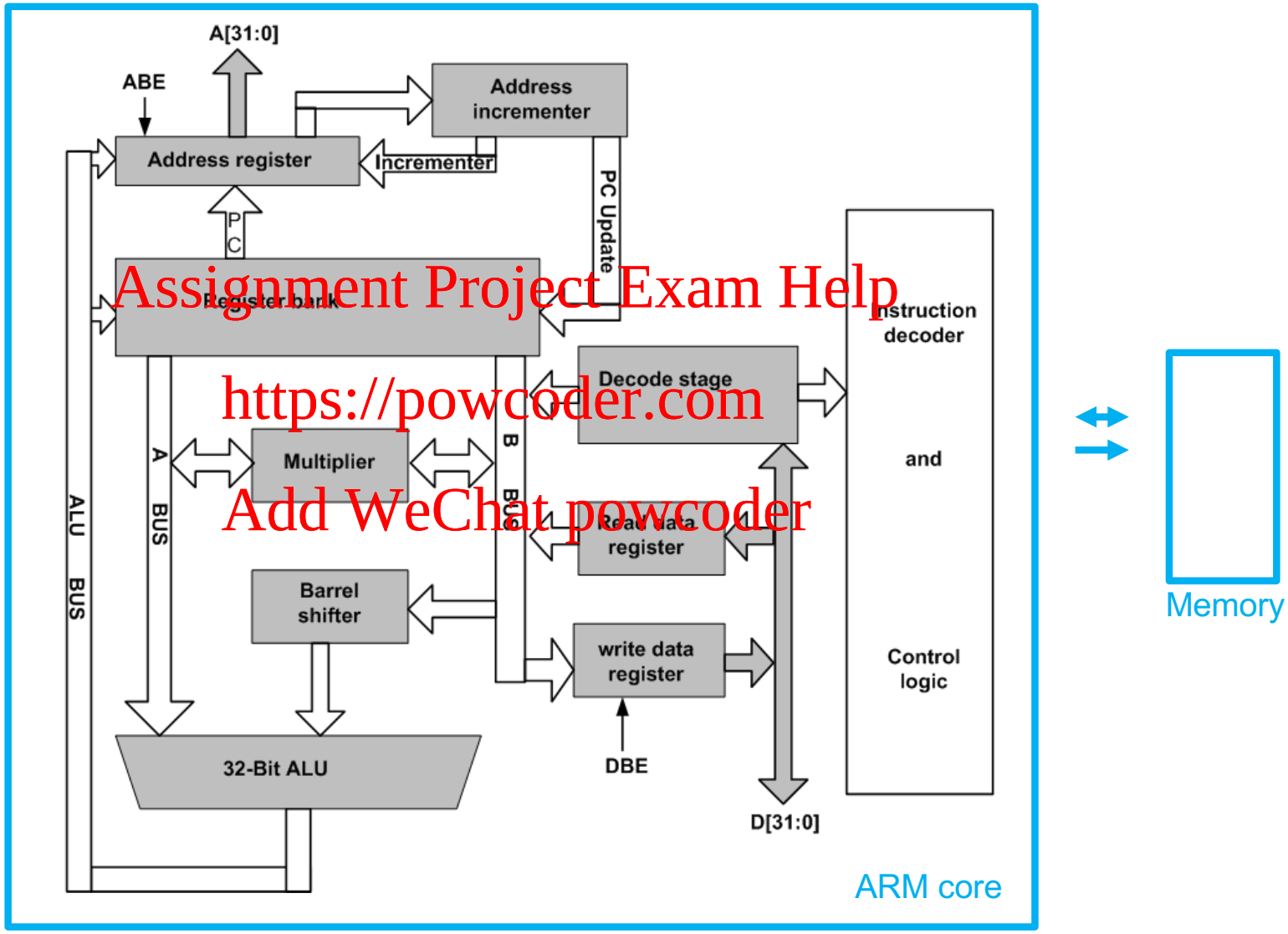
- A description of the ARM7 processor in programmer's perspective
  1. Internal structure: data & control paths
  2. Features available: e.g. what registers are accessible and when
  3. Exception handling: e.g. how the processor responds to an invalid instruction

- You need to know about the **programmer's model** and the **instruction set** of the underlying architecture to start writing assembly programs.

ARM7TDMI processor core

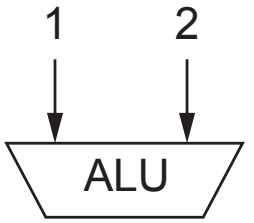


ARM7TDMI processor core



ARM7TDMI instructions

- 1st input always comes from the register bank
- 2nd input can be:
  - Register
  - Immediate operand (comes along with the instruction)
  - A shifted register value (by barrel shifter)
- Result goes to a register
- Examples:



Output reg Input-1 reg

```
ADD R1, R1, #0x5
```

Input-2, an immediate operand

Output reg Input-1 reg

```
ADD R1, R2, R3, LSL #3
```

Input-2, a shifted reg

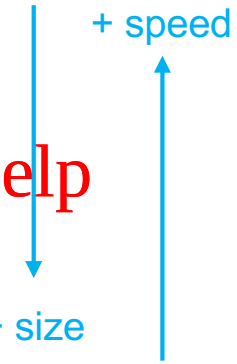
Invalid: cannot shift ALU's input-1

```
ADD R1, R2, LSL #2, R3
```

Memory hierarchy

- Memory hierarchy

	Speed	Size
Registers	A few ns	128 bytes
On-chip Cache	Ten ns	8-32 Kbytes
2nd Cache	A few tens of ns	Hundreds of Kbytes
Main Memory	100ns	Mega bytes
Virtual memory (Hard disk)	tens of milliseconds	100 Gbytes



- Low-level (assembly) programming involves accessing registers and main memory.
- Caches are managed automatically by the hardware.
- Virtual memory is handled by the operating system.

Data types

- Basic element is a binary digit (a bit).
- Bits are organized into:
  - Byte - 8 bits
  - Halfword - 16 bits or 2 bytes
  - Word - 32 bits or 4 bytes
- ARM instructions are 32 bits wide.
- Data is typically handled at word, halfword, and/or byte levels.

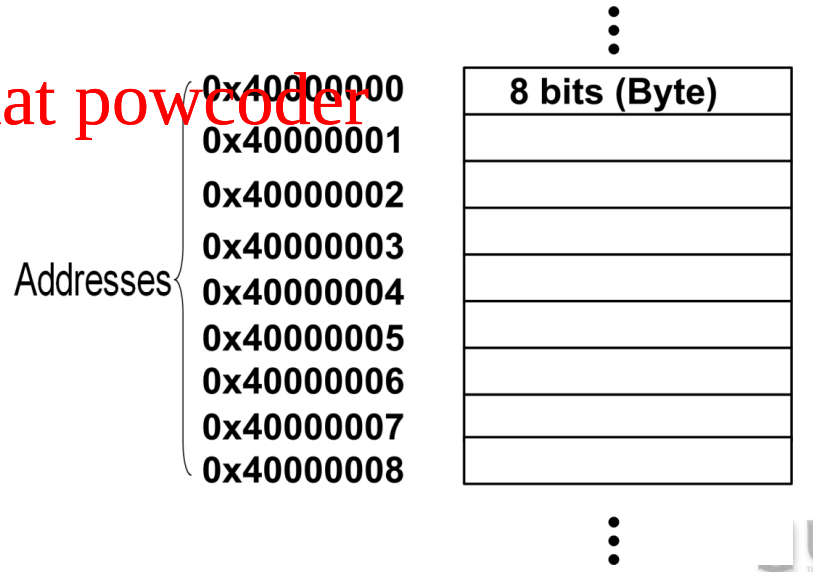
Reading or writing data at **word** level must occur at **word-aligned** memory addresses, e.g. at 0xXXXXXXX0, 0xXXXXXXX4, 0xXXXXXXX8, 0xXXXXXXXC.

Reading or writing data at **halfword** level must occur at **halfword-aligned** memory addresses, e.g. at 0xXXXXXXX0, 0xXXXXXXX2.

Main memory

- Organised into groups (e.g. 8 bits).
- Each memory location has an address.
- N-bit address bus ⇒ 2^N address space.

- ARM7TDMI has 32-bit address bus ⇒ 2^{32} = 4GB of memory space.
- ARM memory contents are 8 bits wide.



Remember for ARM:

- Memory contents are byte oriented.
- Address bus and data bus are 32-bit wide.

Main memory

- Access is limited to **load** and **store** operations, between a memory location and a CPU register.
- Direct memory content manipulation (e.g. adding two variables in the memory) is not permitted. This is known as a **load-store architecture**.
- Some CISC (complex instruction set computer) processors have instructions to directly manipulate contents in the memory. ARM is RISC.

- Example: A+B = C, where A, B and C are variables in memory:
  1. Load A and B from memory into register bank, say R0 and R1.
  2. Perform the add operation R0 + R1, with result written to, say, R3.
  3. Store the result C (in R3) back to memory.

Processor modes

- ARM7TDMI has seven processor modes:
  - 6 privileged modes
  - 1 unprivileged mode.

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt (SWI) instruction is executed	Privileged modes/ Exception modes
FIQ	Entered when a high priority (Fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most applications/OS tasks run	Unprivileged

- Why have modes?

Registers

- Basic storage unit of the data path.
- 32 bits wide (1 word or 4 bytes).
- 37 physical registers:
  - 30 **general purpose registers**
  - 6 **status registers**
  - 1 **program counter** (PC)
- During mode changed, some of the registers are swapped with a set of physically different registers dedicated to the new mode.
  At any given time, the programmer can access 15 general purpose registers (r0, r1, ..., r14), program counter (PC or r15), and one/two status registers.
- USER and SYSTEM share the same register set.

Registers

General purpose registers:

Mode					
User/System	Supervisor	Abort	Undefined	Interrupt -IRQ	Fast Interrupt - FIQ
R0 , A1	R0	R0	R0	R0	R0
R1, A2	R1	R1	R1	R1	R1
R2, A3	R2	R2	R2	R2	R2
R3, A4	R3	R3	R3	R3	R3
R4, V1	R4	R4	R4	R4	R4
R5, V2	R5	R5	R5	R5	R5
R6, V3	R6	R6	R6	R6	R6
R7,V4	R7	R7	R7	R7	R7
R8,V5	R8	R8	R8	R8	R8 _FIQ
R9, V6	R9	R9	R9	R9	R9_ FIQ
R10,V7	R10	R10	R10	R10	R10_ FIQ
R11, V8	R11	R11	R11	R11	R11_ FIQ
R12, ip	R12	R12	R12	R12	R12_ FIQ
R13, sp	R13_ SVC	R13_ ABORT	R13 _UNDEF	R13_ IRQ	R13_ FIQ
R14, lr	R14 _SVC	R14_ ABORT	R14 _UNDEF	R14_ IRQ	R14_ FIQ

Registers

Program counter (PC, R15)

- Accessible in all modes.
- Points to the instruction being fetched from memory.
- Incremented by 4 after each instruction fetch.

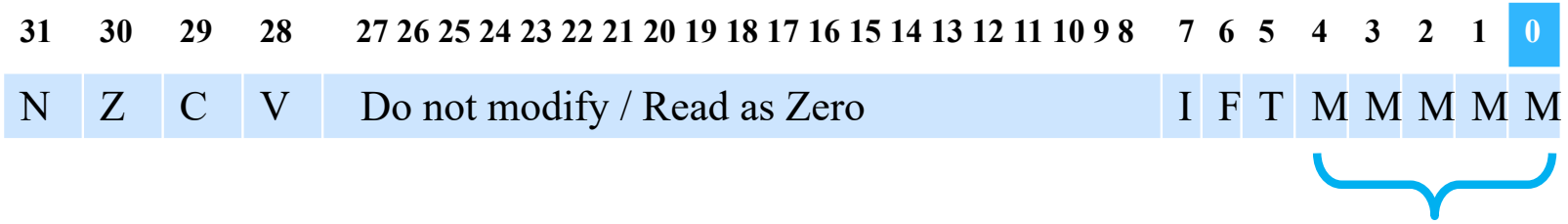
Mode					
User/System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)

Current Program Status Register (CPSR)

- Visible in all modes.
- Stores information on the current processor status.

Mode					
User/System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR _SVC	SPSR_ABORT	SPSR _UNDEF	SPSR_ IRQ	SPSR_ FIQ

Program status registers

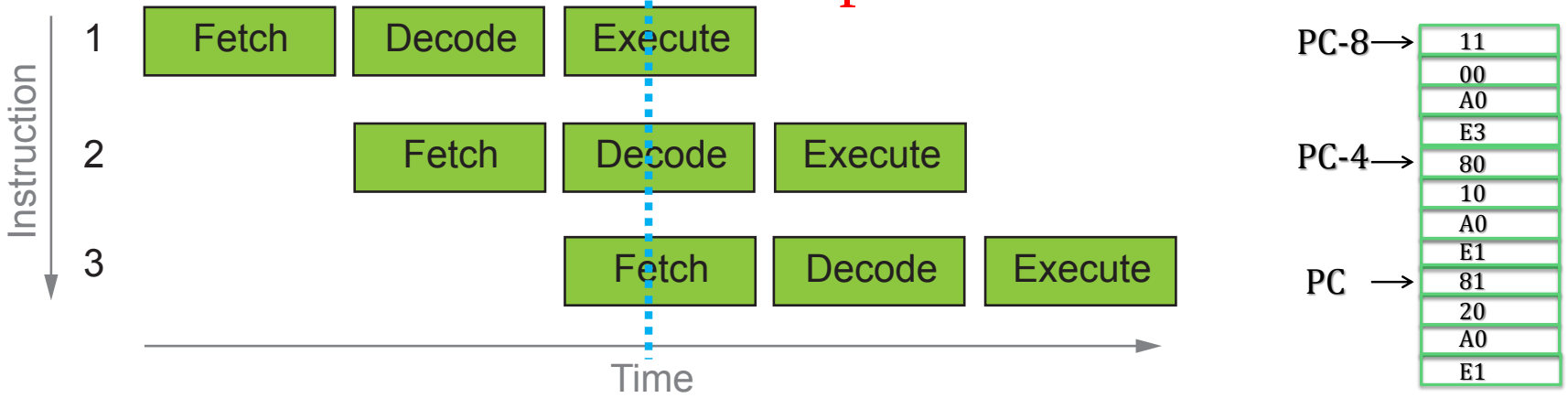


- CPSR and SPSR have the same format.
- Conditional flag bits [31:28]
  - N: negative
  - Z: zero
  - C: carry out
  - V: overflow
- I: disable IRQs
- F: disable FIQs
- T: ARM Thumb instructions (0 for ARM)
- Mode bits [4:0]

The Mode bits	
PSR[4:0]	Mode
10000	User mode
10001	FIQ mode
10010	IRQ mode
10011	Supervisor mode
10111	Abort mode
11011	Undefined mode
11111	System mode

Program counter & 3-stage pipeline

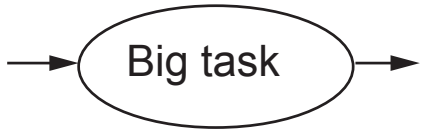
- Dividing a given task into a number of sub-tasks of lower complexity that can be performed in parallel
  - Increases real-time throughput.
- ARM7TDMI is a 3-stage pipelined architecture: **Fetch**, **Decode**, and **Execute**.
- During cycle *i*, instruction pointed to by PC is fetched, while the instruction fetched during cycle *i* - 1 is decoded and the instruction decoded during cycle *i* -1 (i.e. fetched during cycle *i* - 2) is being executed.
- Upon each fetch, PC is automatically incremented by 4.



Pipeline

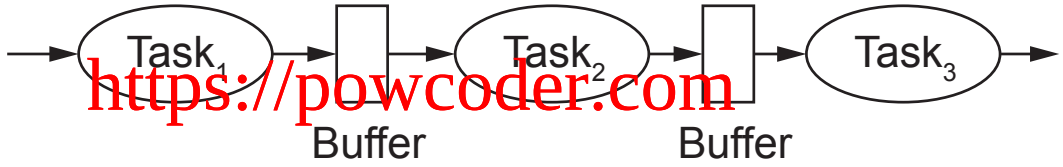
- Exploiting hardware parallelism for increased speed.

Sequential system
(one big task)



$$F = \frac{1}{T}$$

Parallel system
(small tasks)



$T_1, T_2, T_3 \ll T$

$$F = \frac{1}{\max(T_1, T_2, T_3)}$$

- Newer ARMs have 5-stage pipelines.

ARM instruction set

- ARM instructions are 32 bits.
- Instruction types:
  1. Data processing — MOV, ADD, SUB, ADC, RSB, RSC, MUL,...
  2. Data transfer — LDR, STR
  3. Control flow — B, BL
  4. Special — MRS, MSR,...
- Instructions have:
  - Opcode — specifies the operation
  - Operand — can be immediate, register, shifted register
- ARM instructions must align to 4-byte boundaries.

Assembly code format

- General structure for each line of assembly code:
  <label> <instruction / directive / pseudo-instruction> {;comment}

- **Instructions** are actual ARM instructions.
- **Directives** are instructions for the assembler (i.e. not part of the machine instructions).
- **Pseudo-instructions** are “fake” instructions to ease programming. Converted to real instructions during assembly.
- **Labels** are memory addresses that can be used like pointers. Mapped to physical addresses by the Linker.

Assembly examples

For readability: use spaces, not tab character, in your lab assembly code. Comment your code.

Assembler directives

- Constants in ARM assembly code:
  1. Numeric
     - If you don't specify: it is decimal, e.g. 255
     - Hexadecimal: e.g. 0x255
     - Any other base: n_xxxx. n is the base (from 2 to 9), xxxx is the number, e.g. 2_10101010
  2. Character: 'A'
  3. String: "Hello World", (not null-terminated)

- Examples:

```
MOV R0, #2_10101101
MOV R2, #'B'
```

- Directives, register names, assembly mnemonics are not case sensitive.
- Precede each line of instruction with space(s). Labels have no preceding space.

Assembler directives - AREA

- Defines a block of data or code within your source file.
- Syntax:
  AREA <section_name> {,attribute} {,attribute} …
  - section_name is the name of the data or code section. If starting with a digit, it has to be enclosed in bars, e.g. |1_test|
  - Optional attributes can be
    - CODE — a section of code, typically declared as read-only.
    - DATA — a section of data, can be declared as read-write.
    - READONLY — declares that the section cannot modified.
    - READWRITE — declares that the section can be modified.
    - ALIGN=expr — aligns the section into a 2^{expr}-byte boundary.

Assembler directives - EQU

- Giving meaningful names to numeric constants in the program. Similar to #define in C.
- Syntax:
  <name> EQU <expr>
  - name: symbolic name assigned to the constant
  - expr can be:
    - a 32-bit user defined constant
    - an absolute memory address
    - a relative memory address
- Examples:

```
SRAM_BASE   EQU   0x40000000
table       EQU   label+8     ; assigns the program-relative 32-bit
                              ; address label+8 to symbol table
```

Assembler directives - ENTRY

- Declares an entry point to the program, which must have at least one ENTRY.
- In projects with multiple source files, each source will have an ENTRY directive.
- A single source file should not have more than one ENTRY directive.
- Example:

```
AREA testprog, CODE, READONLY
ENTRY
```

Assembler directives - DCB, DCW, DCD

- Used to declare initial run-time content in the memory at byte-, halfword-, and word-level.

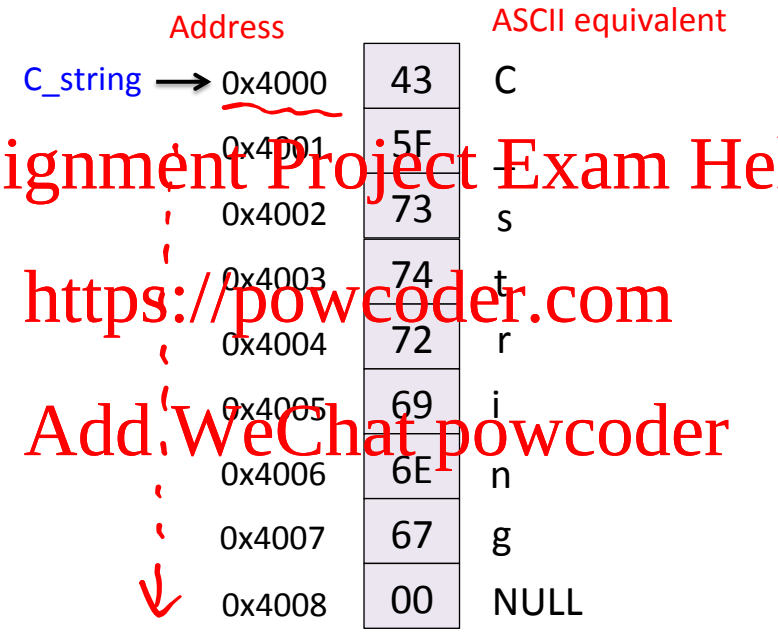
- DCB
  - Allocates and define run-time memory content at byte level.
  - Syntax:
    {label} DCB expr {,expr}...
    - label: memory address of the starting byte.
    - expr: either a numeric expression that evaluates to an integer in -128 to 255, or a string. For strings, the characters are stored in consecutive locations in memory.

  - Explicit memory alignment may be needed using the ALIGN directive.

Assembler directives - DCB, DCW, DCD

- DCB example:

```
C_string  DCB  "C_string",0    ; null-terminated string const
```



Assembler directives - DCB, DCW, DCD

- DCW
  - Defines run-time memory content at halfword level.
  - Syntax:
    {label} DCW{U} expr {,expr}
    - label: memory address of the starting byte of the halfword declared by DCW.
    - Optional U discards alignment, otherwise aligned to 2-byte boundaries.
    - expr: a numeric expression that evaluates to an integer in the range -32768 to 65535.
  - Example:

```
coeff    DCW    0xFE38, 0xDBE5 ; defines two halfwords
```

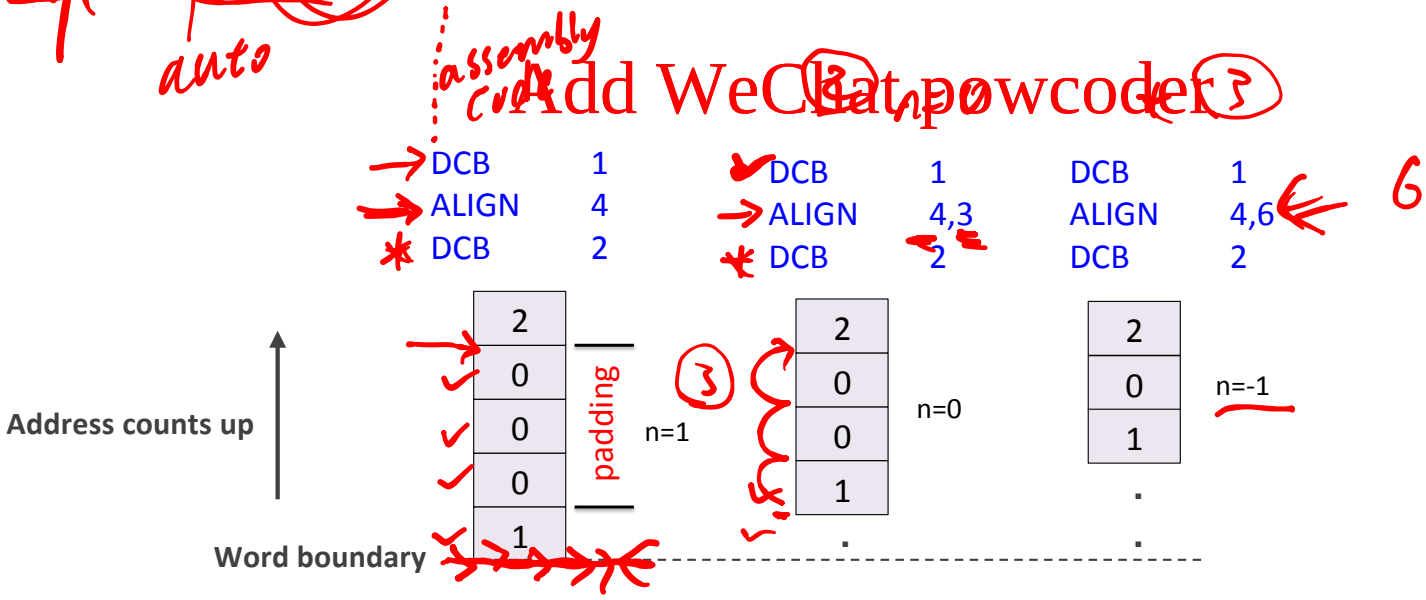
Assembler directives - DCB, DCW, DCD

- DCD
  - Defines run-time memory content at word level.
  - Syntax:
    {label} DCD{U} expr {,expr} …
    - label: memory address pointing to the starting byte of the word declared by DCD.
    - Optional U discards alignment. Otherwise aligned to 4-byte boundaries. That is, DCD might insert up to 3 bytes of padding.
    - expr: is a numeric expression or program relative address.
  - Example:

```
coeff1    DCD    1,5,20
data1     DCD    data2+4
```

Assembler directives - ALIGN

- Aligns the current location in memory to a specified boundary by padding with zeros.
- Syntax:
  ALIGN {expr{,offset}}
  - expr numeric expression evaluating to any power of two from 2^0 to 2^{31}.
  - offset any numeric expression
- Aligned to the next lowest address of the form:
  offset + *n* × *expr*, where $n \in Z$ is **automatically selected** to minimize padding.



Assembler directives - SPACE & END

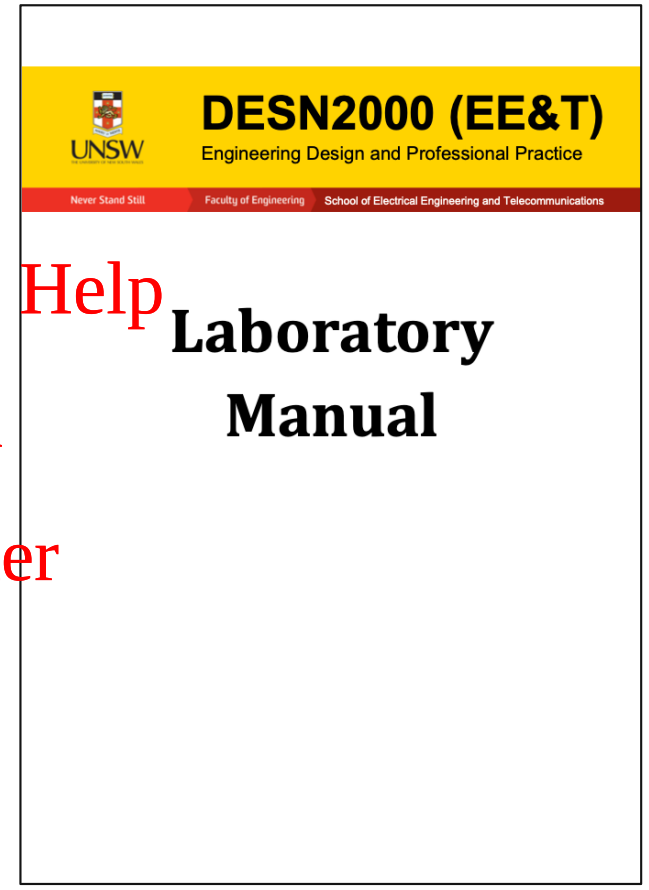
- SPACE allocates a block of memory initialized to zero.
- Syntax:
  <label> SPACE <expr>
  - expr: specifies the number of bytes to reserved as zero.
- ALIGN directive may be used to align any code following the SPACE directive.
- Example:

```
data SPACE 255
```

- END declares the end of the source file.

Lab logistics

- From Moodle: (1) **lab manual** and (2) **lab files**.
- Working in pairs, pick your team in Moodle.
- Get marks for completing each lab's checkpoint(s).



- Remote students:
  - Access lab computer via Teams invite sent to you: (1) desktop control, (2) camera feed of dev. board, (3) web-enabled oscilloscope.
  - Use online Queue Sheet (linked in Teams) to get marked by demonstrator.
  - Communicate with demonstrators via Teams.

- **Do most coding at home. Transfer the files to lab PC using UNSW's OneDrive via web browser on the lab PC.**

This week

- Introduction to embedded systems
- ARM7TDMI programmer's model
- ARM instructions and tools
- Assembly language examples
- Assembler directives and operators

In Moodle:

- **Week 1 Moodle exercise.**
- **Pick Lab team and Workshop team (in Moodle, starting Thurs.)**
- **Start working on Lab 1, esp. if you are in the Monday labs (Due: end of your 3-hr lab)**