

Digital Electronics and Microprocessor Systems (ELEC211)

Dave McIntosh and **Valerio Selis**

dmc@liverpool.ac.uk
[**V.Selis@liverpool.ac.uk**](mailto:V.Selis@liverpool.ac.uk)

Outline

- Recap
- Flags
 - N, V, C, Z
- Conditional execution
- Unconditional execution
- Branches
- Uses of flags

Recap

Instructions: from machine code to mnemonics

- E.g. 0x2372 \Rightarrow MOVS r3, #114

Simple instructions:

- MOVS rd, rm

MOVS rd, #imm

Arithmetic instructions:

- ADDS rz, ry, rx
- SUBS rz, ry, rx
- RSBS rz, ry, #0

ADDS rz, ry, #imm

SUBS rz, ry, #imm

MULS rx, ry, rx

Logical instructions:

- ANDS ry, rx
- EORS ry, rx

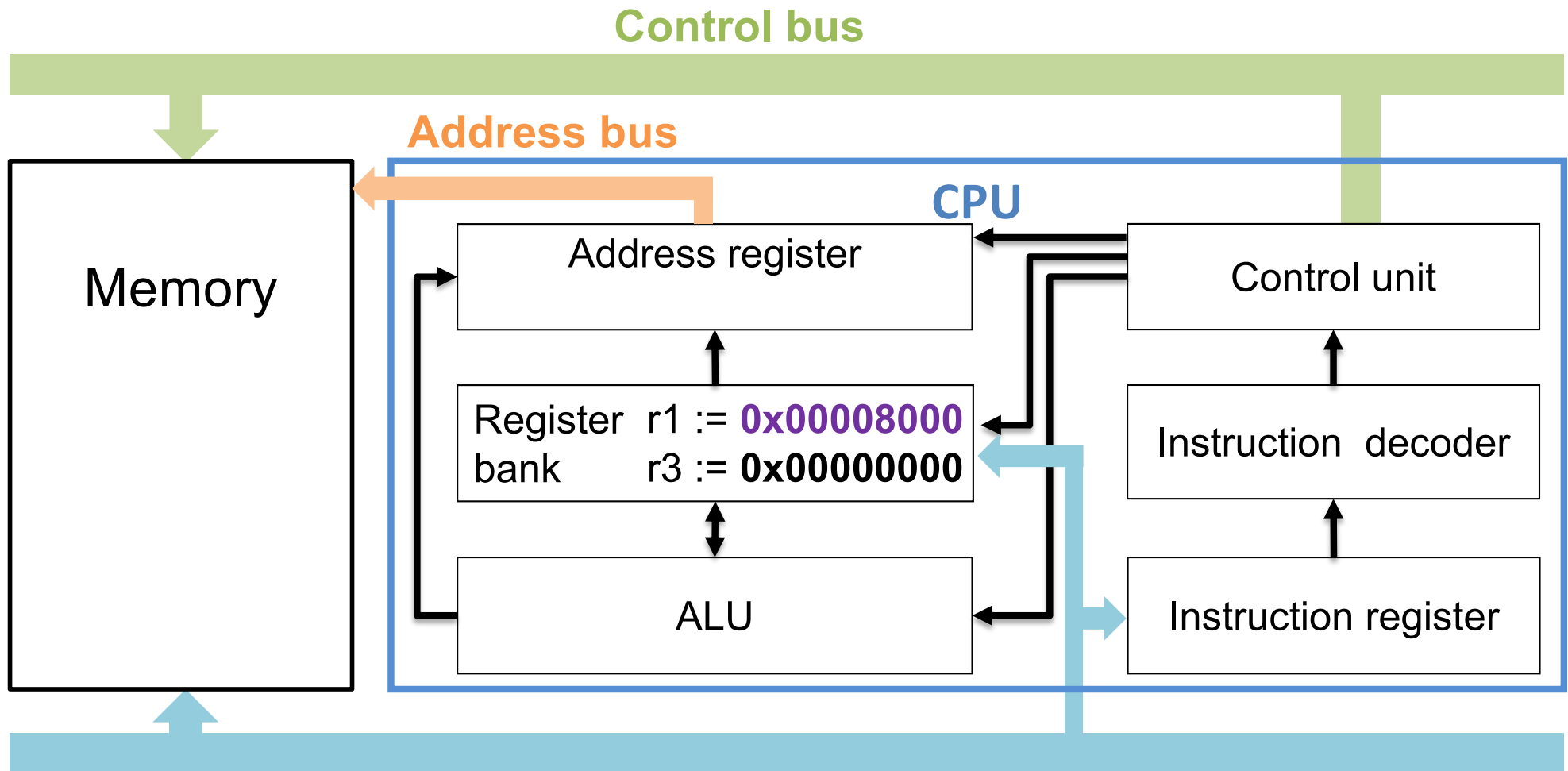
ORRS ry, rx

BICS ry, rx

Recap – Load instruction

LDR r3, [r1]

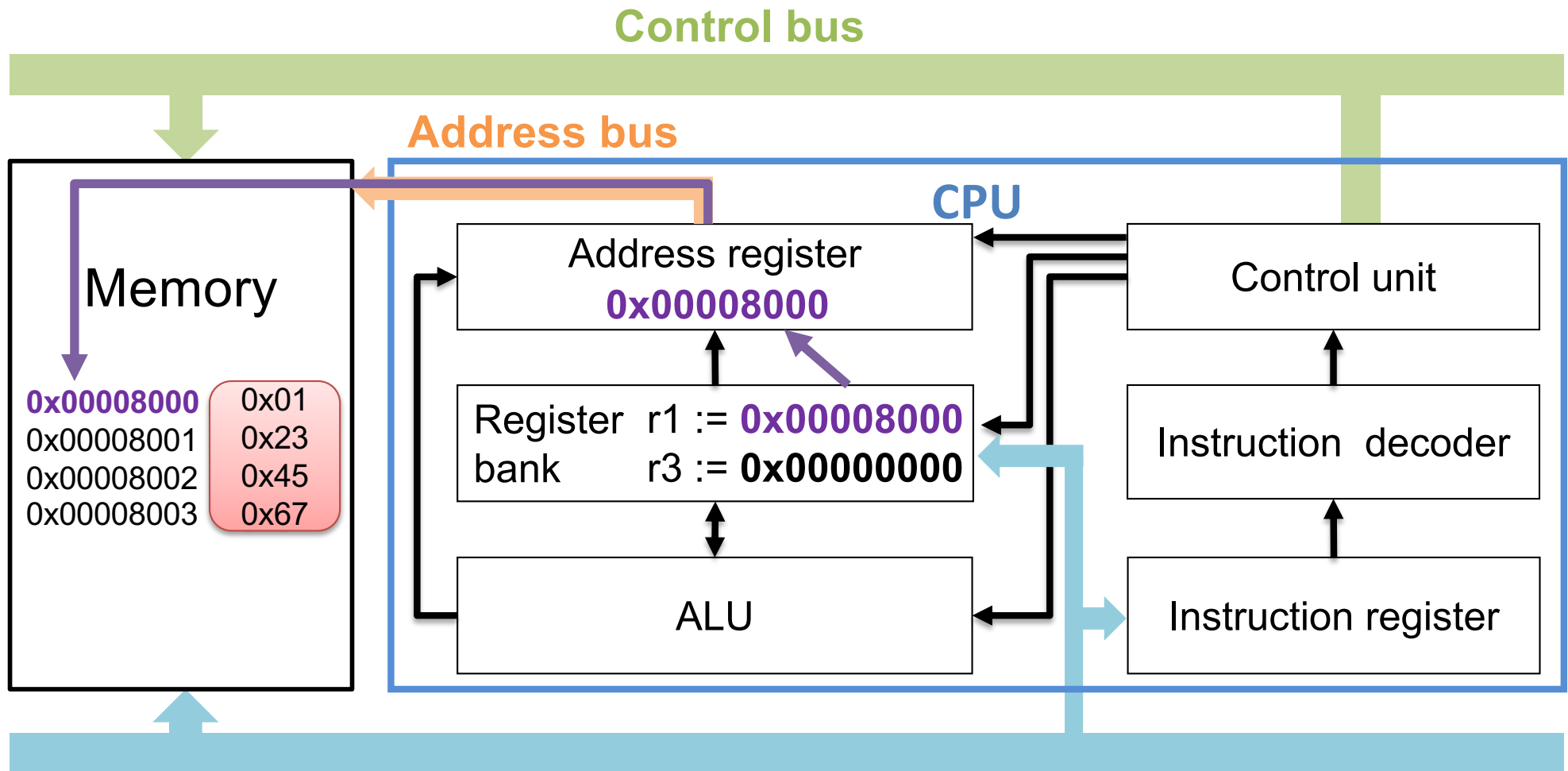
Before Execution



Recap – Load instruction

LDR r3, [r1]

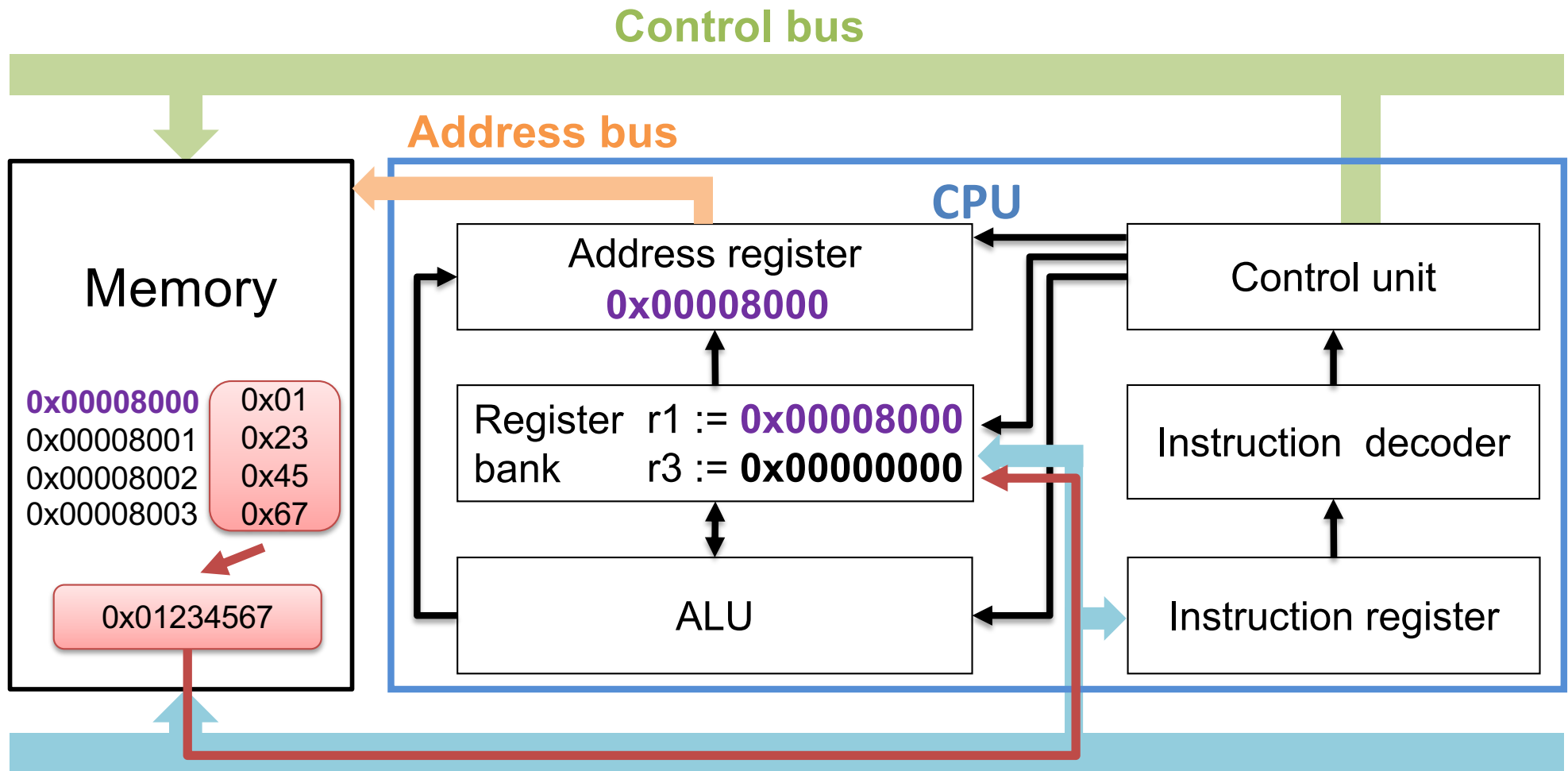
During Execution



Recap – Load instruction

LDR r3, [r1]

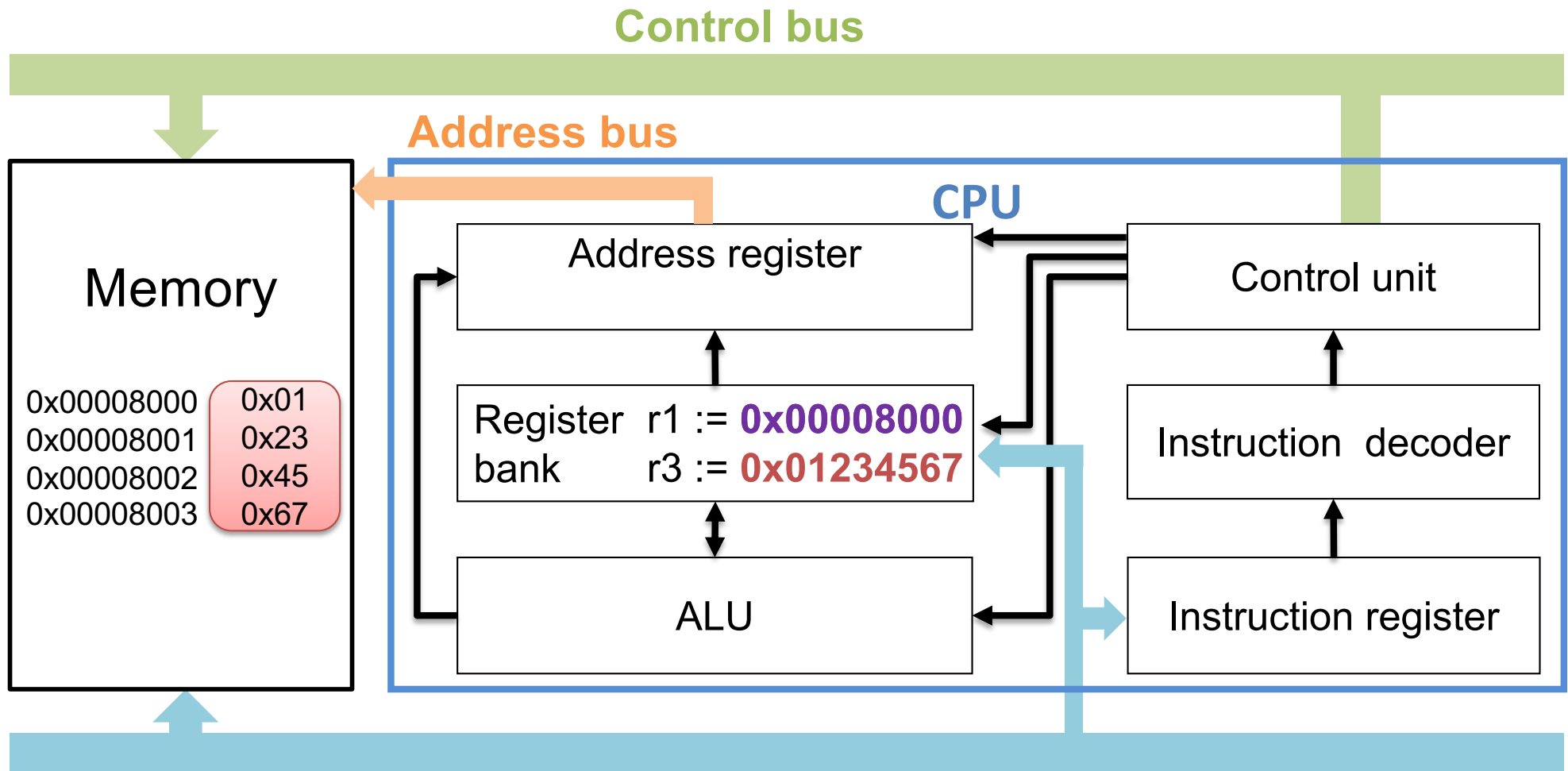
During Execution



Recap – Load instruction

LDR r3, [r1]

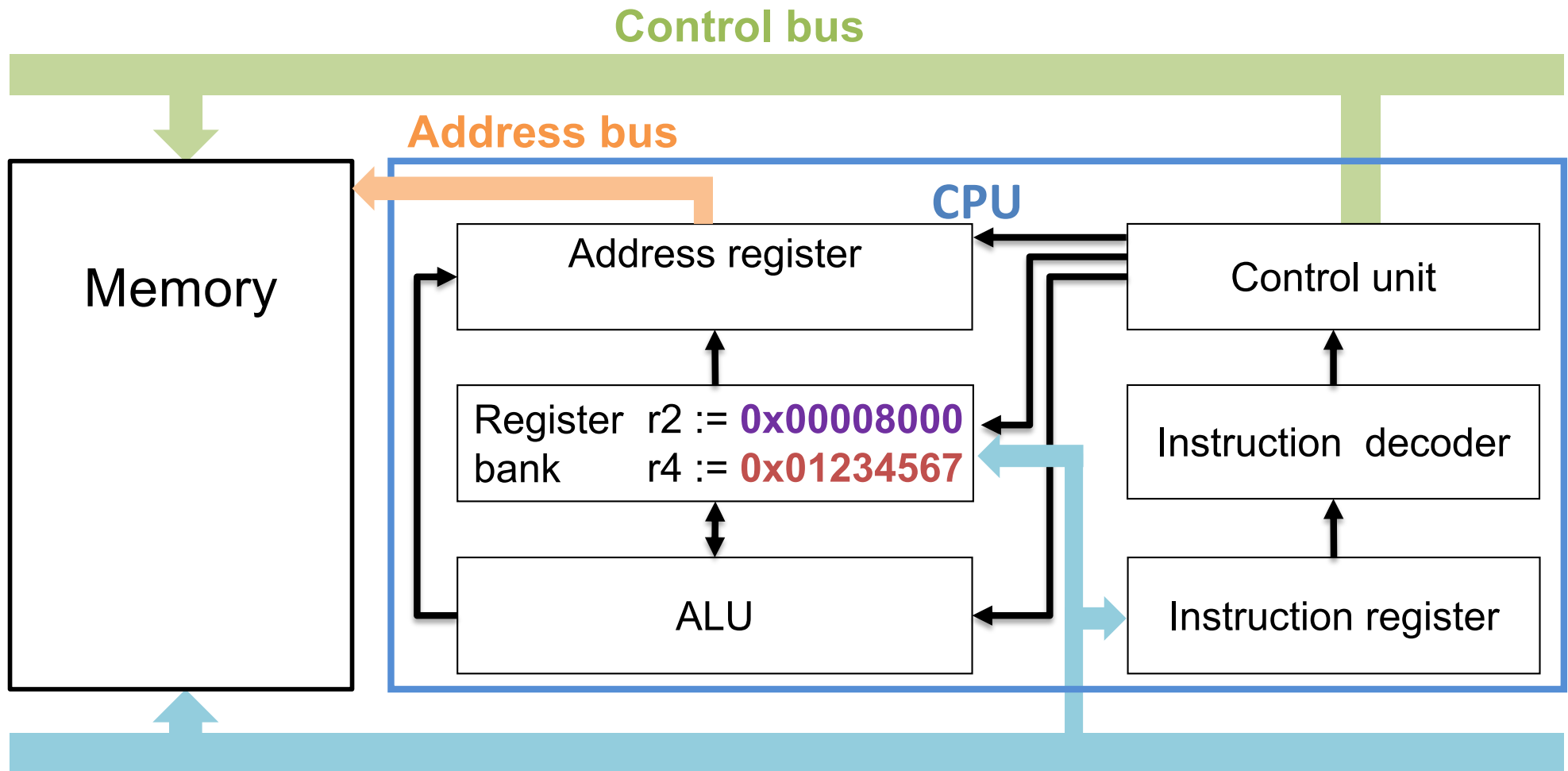
After Execution



Recap – Store instruction

STR r4, [r2]

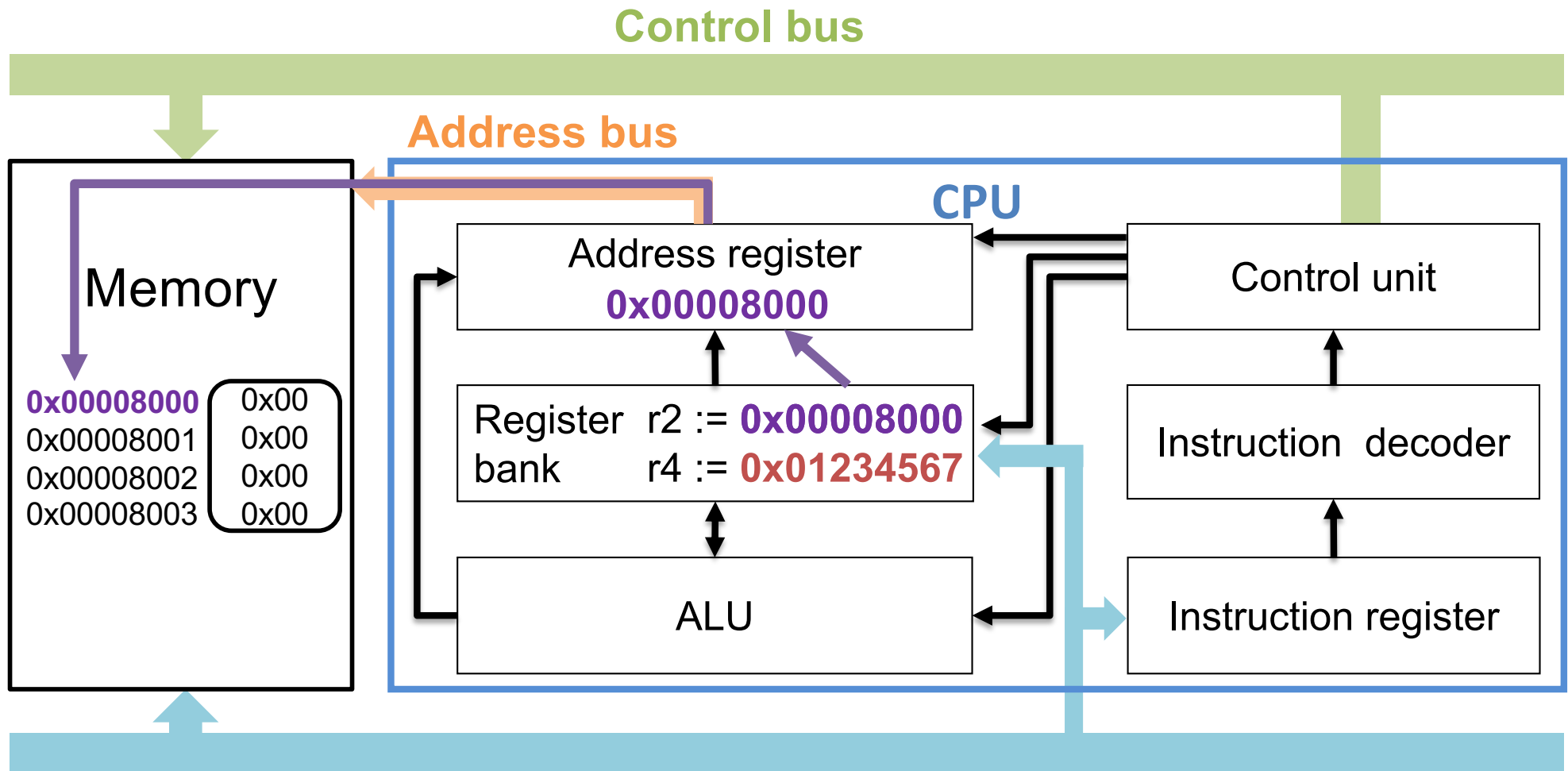
Before Execution



Recap – Store instruction

STR r4, [r2]

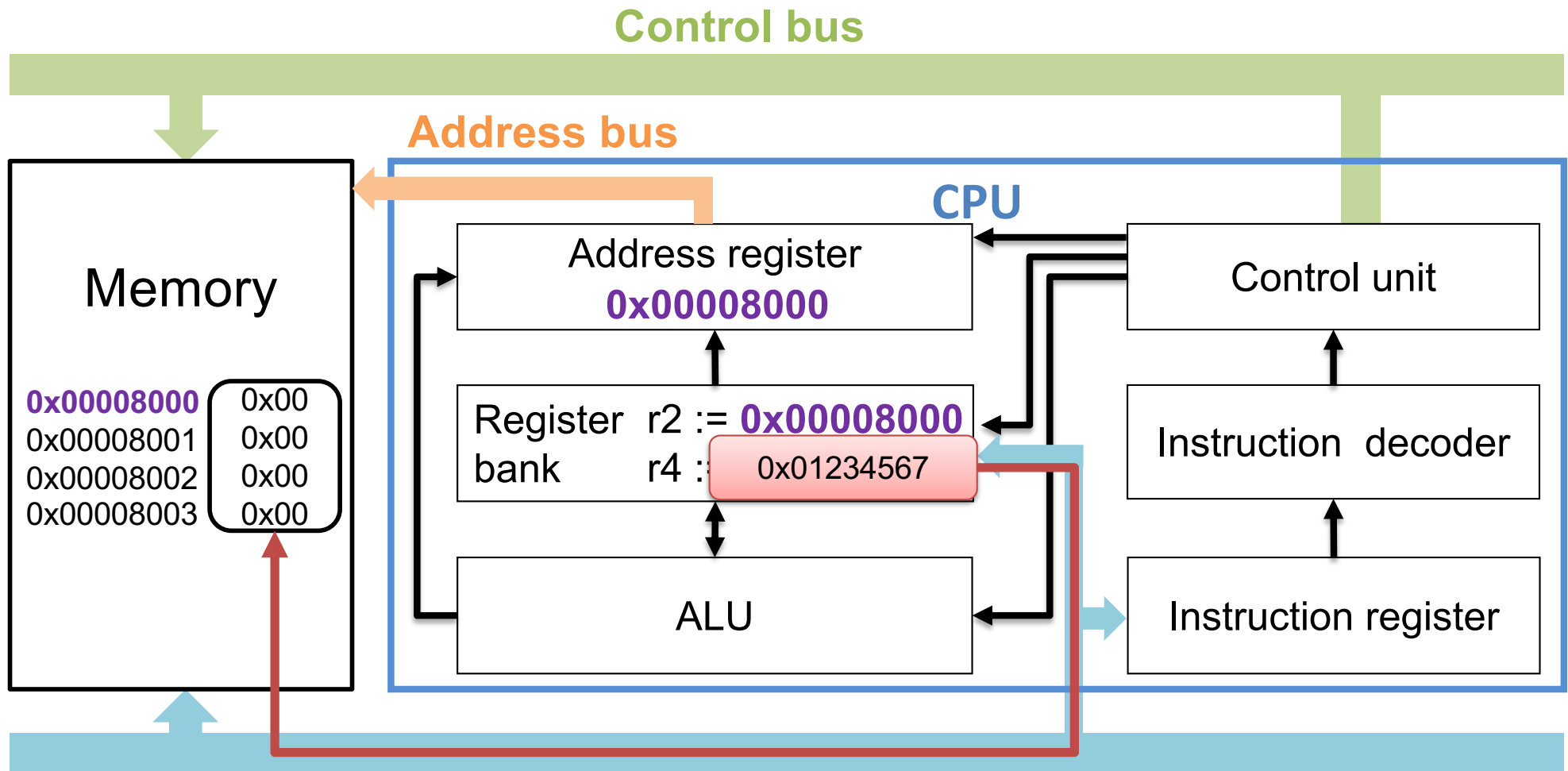
During Execution



Recap – Store instruction

STR r4, [r2]

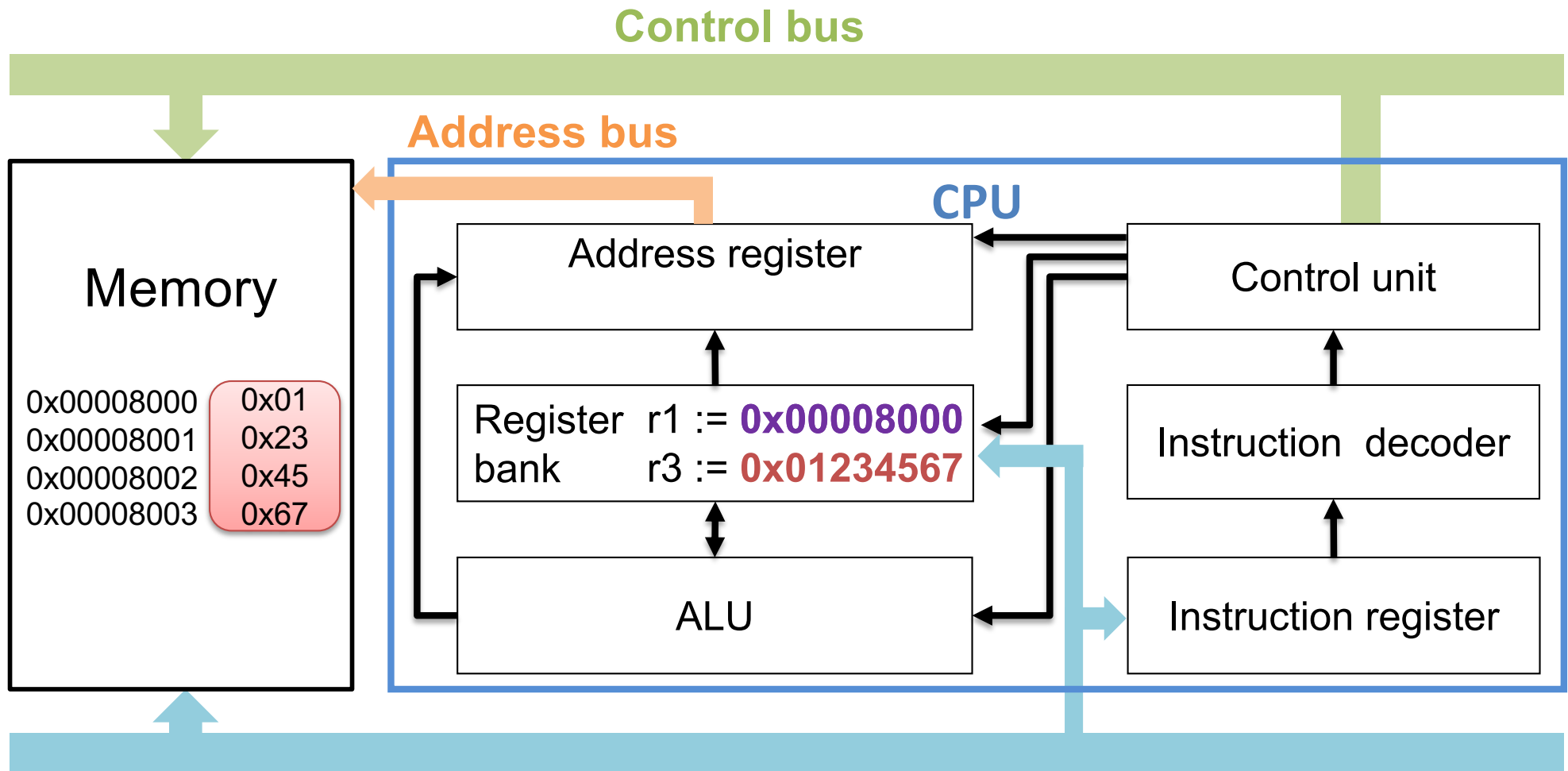
During Execution



Recap – Store instruction

STR r4, [r2]

After Execution



Fun with Flags



Flags

Flags can be used to give signals

- Something either has or has not happened.

Flags are essentially 1 bit memory devices.

- Microprocessors use flags to signal what has happened
- Different microprocessors have different flags

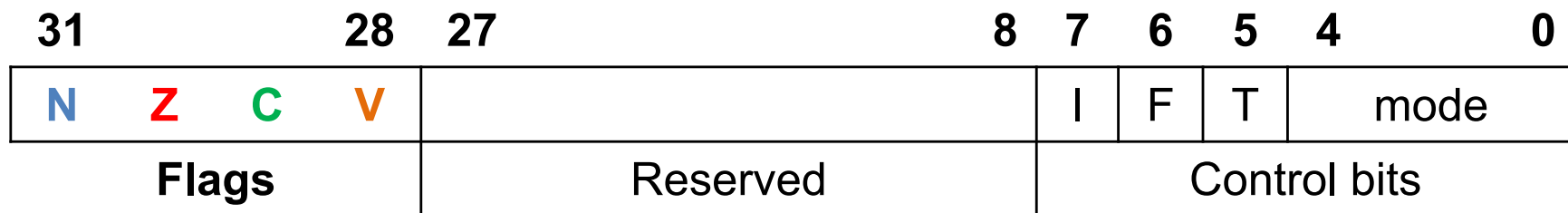
The ARM Cortex M0 processor has four flags that are commonly found in the majority of processors:

- Zero flag (Z), Negative flag (N), Carry flag (C) and Overflow flag (V)

Flags

In the ARM cortex M0 these four flags are stored in a register called Current Program Status Register (CPSR).

- They indicate what happened as a consequence of executing instructions



Negative flag (**N**) ►► The result is **N**egative.

Zero flag (**Z**) ►► The result is **Z**ero.

Carry flag (**C**) ►► A **C**arry has been generated in the ALU

Overflow flag (**V**) ►► The instruction has produced o**V**erflow

Flags

When ARM executes most instructions it automatically generates 4 signals (N, Z, V, C)

- N, Z, V, C are only copied into the corresponding CPSR bits when the programmer deliberately indicates so

All instructions except the branch instruction (B) include the optional item {S}.

- This optional item **S** controls the action “set flags”

For example, the following instructions set or clear the flags as appropriate:

MOVS, ADDS, SUBS, RSBS, MULS, ANDS, EORS,
ORRS, BICS

Flags – example ADD

ADD, which does not modify the values of the N, Z, C and V bits in the CPSR registers.

Before Execution				
CPSR	0	0	0	0
	N	Z	C	V
	Flags			

ADD r3, r2, r5

r2 := 0x C F 0 E C 5 3 5 +

r5 := 0x 3 7 4 2 4 1 1 4 =

1 0 6 5 1 0 6 4 9 → r3

After Execution				
CPSR	0	0	0	0
	N	Z	C	V
	Flags			

Flags – example ADDS

ADDS, which does modify the values of the N, Z, C and V bits in the CPSR registers.

Before Execution				
CPSR	0	0	0	0
	N	Z	C	V
	Flags			

ADDS r3, r2, r5

r2 := 0x C F 0 E C 5 3 5 +

r5 := 0x 3 7 4 2 4 1 1 4 =

1 0 6 5 1 0 6 4 9 → r3

After Execution				
CPSR	0	0	1	0
	N	Z	C	V
	Flags			

Flags – example ADDS

The result of the addition cannot be stored in a register as its value was higher than $2^{32}-1$.

- There was a carry out of '1'.
- The ADDS instruction **always** moves the value of the carry bit into the CPSR register.
- 1 if the result is greater than $2^{32}-1$ or 0 if it is lower.

ADDS r4, r0, r1

r0 := 0x 0 F 0 E C 5 3 5 +

r1 := 0x 0 7 4 2 4 1 1 4 =

0 1 6 5 1 0 6 4 9

↓
r4

Before Execution

0	0	1	0
N	Z	C	V

Flags

After Execution

0	0	0	0
N	Z	C	V

Flags

Flags – example Z flag

If an instruction such as a subtraction produces a result which is 0x00000000 then the zero flag would be set to 1.

Instructions that would set the zero flag are:

`MOVS r5, #0` - move zero into register r5 and

`SUBS r2, r1, r1` - subtract value in r1 from itself and put the difference (zero) in r2.

Flags

Flags are used in two ways:

- The main use of flags is to determine if a branch instruction is executed or not, called **conditional execution**.
- The carry flag can also be used in some arithmetic instructions as an additional value (we will return to this later).

Conditional execution

Conditional execution means that a branch instruction either **does** or **does not** execute depending upon the state of the flags (or 'condition codes').

The mnemonic for a branch instruction is **B** and conditional execution is indicated by the addition of two letters in the mnemonic.

- Example: **BCS** means execute the **B** Branch only if the **C**arry flag is **S**et.

There are 15 different condition fields which may be appended to (almost) any mnemonic.

Conditional execution

The common ones are:

- EQ: 'equal', it is executed only if the zero flag (Z) is set
- NE: 'not equal', it is executed if the zero flag (Z) is clear
- CS: 'carry set', it is executed if the carry flag (C) is set
- CC: 'carry clear', it is executed if the carry flag (C) is clear
- MI: 'minus', it is executed only if the negative flag (N) is set
- PL: 'plus' or 'positive', executed only if the negative flag (N) is clear
- VS: 'overflow', executed if V is set
- VC: 'no overflow', executed if V is clear
- AL: 'always', execute always unconditionally (default if no condition field is specified).

Branches

‘Normal’ CPU operation follows a sequence:

- Instruction Fetch
 - Increment the Program Counter (PC).
- Instruction Decode
- Instruction Execution

A branch instruction breaks this sequence:

- Loads the Program Counter with a new value.
- The processor does a branch to a new memory position rather than the next one


Branches

Two types of branches:

- Unconditional branch:
 - Always executes and it reloads the program counter with a new value.
- Conditional branch:
 - Depends on the condition (status of the flags):
 - It will branch
 - Or it will continue normal execution

Unconditional branch – example

Consider the following example programme:



<u>Memory address</u>	<u>Instruction</u>
0x00000082	EORS r4, r4, r5 ; xor
0x00000084	B 0x00000088 ; branch
0x00000086	ANDS r5, r5, r2 ; and
0x00000088	MOV r8, r6 ; move
0x0000008A	B 0x00000086 ; branch

These instructions are executed in the following order:

EORS, B 0x88, MOV, B 0x86, ANDS, MOV,
B 0x86, ANDS, MOV, B 0x86, ...

Conditional Branches

Unconditional branches are of very little use but conditional branches have many uses.

Any condition field can be used with branch e.g.

BNE 0x08C00000

- Means branch to 0x08C00000 if Not Equal:
 - 1) If **Z flag** is clear executes the instruction at address 0x08C00000, or
 - 2) If **Z flag** is set executes the instruction immediately following the branch



Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

What values are held in r4, r5 and r15 after the execution of the following?

Memory Address Instruction

0x00008000 SUBS r4,r7,r7

0x00008002 BEQ 0x00008006

0x00008004 MOVS r4,#17

0x00008006 ADDS r5,r4,#6

r4=0x00000011, r5=0x00000017, r15=0x00008008

r4=0x00000000, r5=0x00000006, r15=0x00008008

r4=0x00000011, r5=0x00000017, r15=0x00008006

r4=0x00000000, r5=0x00000006, r15=0x00008006

Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

Total Results





Question

What values are held in r4, r5 and r15 after the execution of the following?

Memory Address Instruction

0x00008000 SUBS r4,r7,r7

0x00008002 BEQ 0x00008006

0x00008004 MOVS r4,#17

0x00008006 ADDS r5,r4,#6

r4=0x00000011, r5=0x00000017,
r15=0x00008008

r4=0x00000000, r5=0x00000006,
r15=0x00008008

✓ 0%

r4=0x00000011, r5=0x00000017,
r15=0x00008006

r4=0x00000000, r5=0x00000006,
r15=0x00008006



Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app



Conditional Branches – range

B<cond> <target_address>

- The lower 8 bits of the machine code are used to determine the destination address. It is an offset from the current program counter value.
- The difference between the destination address and the memory address of the branch instruction must be an even number within the range: +254 to -256.

15				12		11		8		7		0			
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	
Opcode				cond				target_address (signed imm 8)							

Unconditional Branches – range

B <target_address>

- The lower 11 bits of the machine code are used to determine the destination address as an offset from the current program counter value.
- The difference between the destination address and the memory address of the unconditional branch instruction must be an even number within the range +2046 to -2048.

15					11	10					0				
1	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X
Opcode					target_address (singed imm 11)										

Branches – mnemonics

In general assembly language programmes are written without any knowledge of the memory address for the corresponding machine code.

So the mnemonic for branch uses '**labels**' rather than actual memory addresses and the assembler programme determines the actual value used.

E.g.

BNE **continue**

SUB r4, r7, r7

continue ADDS r4, r4, #76

Use of the zero flag

The zero flag is commonly used to test if a program '**loop**' has been executed a certain number of times;

For example, if one wished to add together 10 numbers from memory this could be implemented in a loop that repeated 10 times.

A register would be chosen as the '**loop counter**'; during each loop the counter would decrease by 1 and the loop repeats until the counter reaches 0.

Use of the zero flag – example

Programme to add together **10** values from memory 0x00000080 onwards:

	MOVS r0, # 10	; set loop counter
	MOVS r1, #0x080	; set address for data
	LDR r2, [r1]	; load 1st value
loop	ADDS r1, r1, #4	; increment base register
	LDR r3, [r1]	; load next value
	ADD r2, r3	; add r3 to running total in r2
	SUBS r0, #1	; decrement loop counter
	BNE loop	; branch back to loop if zero ; flag is cleared.

zero flag – only bottom 32 bits

The zero flag is set or cleared based only on the 32 bits held in a register.

For example, if **0xF0F0F0F1** is added to **0x0F0F0F0F** then the result using an unlimited number of bits to **0x100000000**:

$$\begin{array}{r} 1111\ 0000\ 1111\ 0000\ 1111\ 0000\ 1111\ 0001 \\ +\ 0000\ 1111\ 0000\ 1111\ 0000\ 1111\ 0000\ 1111 \\ \hline 1\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array}$$

A destination register would only hold the **bottom 32 bits**, 0x00000000, and the Z flag would be set by this addition, ignoring the **1** that is outside the range.

The carry flag

The carry flag is used in many arithmetic and logic instructions. The use of the carry flag is best illustrated by looking at addition.

Consider a processor with 32 bit registers such as the ARM Cortex M0.

If the sum of two numbers is greater than $0xFFFFFFFF$ ($= 4,294,967,295_{10}$) then the sum will have more than 32 bits and it cannot be fitted into a 32 bit register.

The carry flag – example

If r2 holds the value 0xF200F309 and r7 holds the value 0x42002204 and the following instruction is executed:

ADD**S** r5, r2, r7

The sum held in r5 should be

0x13401150D or

1 0011 0100 0000 0001 0001 0101 0000 1101₂

but the register cannot hold the **most significant bit**, therefore, the instruction will set the carry flag and r5 will hold **0x3401150D**.



Question

When poll is active, respond at **PollEv.com/elec211**

Text **ELEC211** to **22333** once to join

If r5 holds the value 0xFFFFFFFF, what happens to the zero and carry flags after each addition in the following programme?

ADDS r4,r5,#0 ;add 0 to r5

ADDS r4,r5,#1 ;add 1 to r5

ADDS r4,r5,#2 ;add 2 to r5

1st: Z and C cleared, 2nd: Z and C cleared, 3rd: Z cleared and C set

1st: Z set and C cleared, 2nd: Z and C set, 3rd: Z cleared and C set

1st: Z and C set, 2nd: Z and C cleared, 3rd: Z set and C cleared

1st : Z and C cleared, 2nd: Z and C set 3rd: Z cleared and C set


Start the presentation to see live content. Still no live content? Install the app or get help at PolleEv.com/app

Total Results





Question

 **Poll locked.** Responses not accepted.

If r5 holds the value 0xFFFFFFFF, what happens to the zero and carry flags after each addition in the following programme?

ADDS r4,r5,#0 ;add 0 to r5

ADDS r4,r5,#1 ;add 1 to r5

ADDS r4,r5,#2 ;add 2 to r5

1st: Z and C cleared, 2nd: Z and C cleared, 3rd: Z cleared and C set

1st: Z set and C cleared, 2nd: Z and C set, 3rd: Z cleared and C set

1st: Z and C set, 2nd: Z and C cleared, 3rd: Z set and C cleared

1st : Z and C cleared, 2nd: Z and C set 3rd: Z cleared and C set

✓ 0%

Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app



Recommended reading

ARM system-on-chip architecture.

- Section 5.3: Conditional execution.
- Section 5.4: Branch and Branch with link (B, BL).

Summary



Flags (N, V, C, Z)

Conditional and unconditional executions

Branches

Uses of flags

Next class?

Wednesday at 12 noon in the
Chadwick building,
Barkla Lecture Theatre
(CHAD-BARKLA)