Follow Steve Furber 'ARM System on a Chip Architecture Lecture Notes

1. Course overview
2. Intro to PICOBLAZE, C and Number systems and Boolean Algebra
3. Course overview with microprocessor MU0 (I)
4. Course overview with microprocessor MU0 (II)
5. Verilog HDL
6. Digital system components using schematics and Verilog
7. Combinational logic standard forms. Karnaugh maps
8. Combinational ccts and configurable logic devices
9. Simple Sequential circuits, flip flops
10. Sequential circuits, counters, registers, memories
11. Non-ideal effects in digital circuits
12. Finite State Machines
13. Design of FSMs
14. Design of FSMs
15. Datapaths

16. An introduction to Processor Design
17. The ARM Architecture
18. ARM Asssembly Language Programming
19. Programming in C
...

ARM history

- 1983 developed by Acorn computers
  - To replace 6502 in BBC computers
  - 4-man VLSI design team
  - Its simplicity comes from the inexperienced team
  - Match the needs for generalized SoC for reasonable power, performance and die size
- 1990 ARM (Advanced RISC Machine), owned by Acorn, Apple and LSI

ARM Ltd

Design and license ARM core design but not fabricate

Why ARM?

- One of the most licensed and thus widespread processor cores in the world
  - Used in PDA, cell phones, multimedia players, handheld game console, digital TV and cameras
  - ARM7: GBA, iPod
  - ARM9: NDS, PSP, Sony Ericsson, BenQ
  - ARM11: Apple iPhone, Nokia N93, N800
  - 75% of 32-bit embedded processors
  - Cortex – beagleboard open hardware
- Used especially in portable devices due to its low power consumption and reasonable performance

http://beagleboard.org/

http://alwaysinnovating.com

ARM LPC2368 Dev. Board in HLABs (*Furturlec)*

ARM LPC2368 Dev. Board (*Furturlec)*

- Includes NXP LPC2368 ARM Microcontroller with a huge 512kb Internal Flash Program Memory
- Operating Speed up to 72 MHz
- Direct In-Circuit Programming via RS232 Connection for Easy Program Updates
- Up to 25 I/O points with easy to connect standard headers
- Full Speed USB 2.0 Port
- Ethernet LAN 10/100Mb Connection for full networking
- Large 58k Data RAM
- 6 Channels 10-Bit A/D
- 1 Channel 10-Bit DAC
- 2 Channels standard CAN network
- Real Time Clock with Battery Back-Up
- SD Card Connector for Data Storage and Transfer
- JTAG Connector for Program Download and Debug
- LCD Connection with Contrast Adjustment
- Load and Reset Button
- On-Board 3.3V Regulator
- Ideal as an Interchangeable Controller for Real-Time Systems

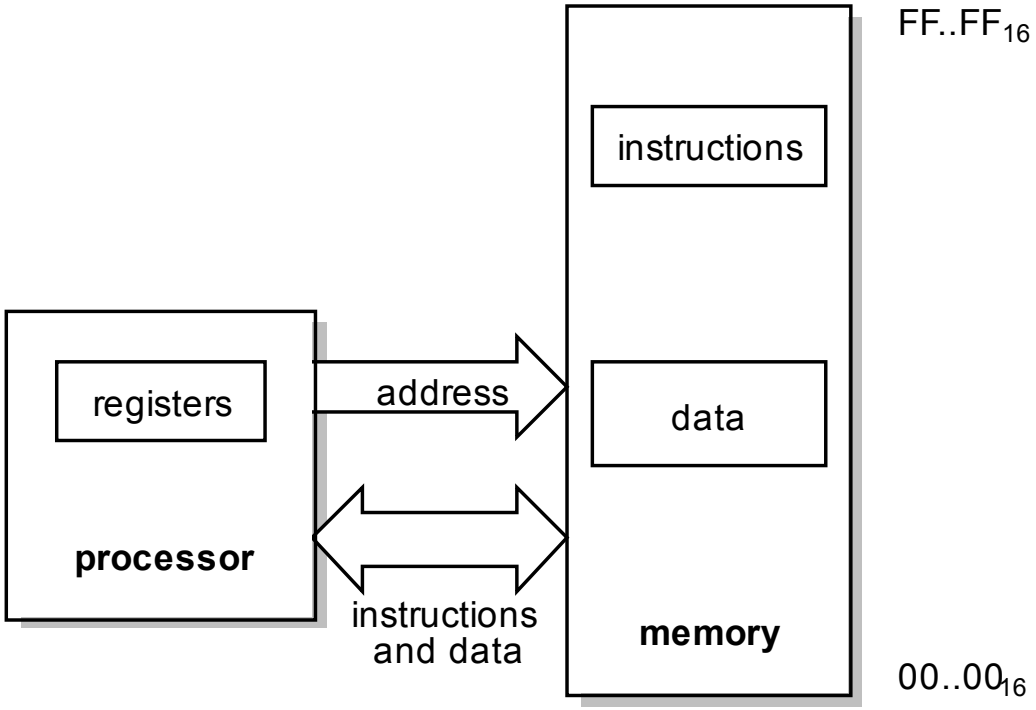
Computers

- All modern-general purpose computers employ the principles of a stored program digital computer (dates to 1940s)
- First implemented SSEM ('Baby') June 1948 Uni Manchester

The Small-Scale Experimental Machine, known as SSEM, or the "Baby", was designed and built at The University of Manchester, and made its first successful run of a program on June 21st 1948. It was the first machine that had all the components now classically regarded as characteristic of the basic computer. Most importantly it was the first computer that could store not only data but any (short!) user program in electronic memory and process it at electronic speed.

- **Most advances in computing due to electronics... but ..**
  - **Computer Architecture = User view: instruction set, visible registers, memory management…**
  - **Computer Organisation: user-invisible – pipeline structure, transparent cache, ...**

The state in a stored-program digital computer



MU0 – A simple microprocessor

- A simple form of processor can be built with Program counter PC
  - Accumulator or working register
  - Arithmetic logic unit
  - Instruction register (IC)
  - Instruction decode and control logic that employs the above components to achieve the desired results from each instruction

- MU0 is a 16 bit machine with a 12 bit address space
- Instructions are 16 bits long with a 4 bit opcode and a 12 bit address word

- **The datapath:** All the components carrying (buses), storing (registers) or processing (alu, mux) bits in parallel form the components of the datapath. **Use the RTL description – actually datapath for us!**

- **The Control Logic:** Everything else such as decode and control use **FSM approach.**

MU0 – Datapath design

Need a guiding principle to limit design possibilities – usually based on clock constraint in microprocessors

- ***Each instruction takes the number of clock cycles equal to the number of memory accesses it must make***

- We assume an instruction starts when the instruction appears in the instruction register. There are generally two steps to execute an instruction

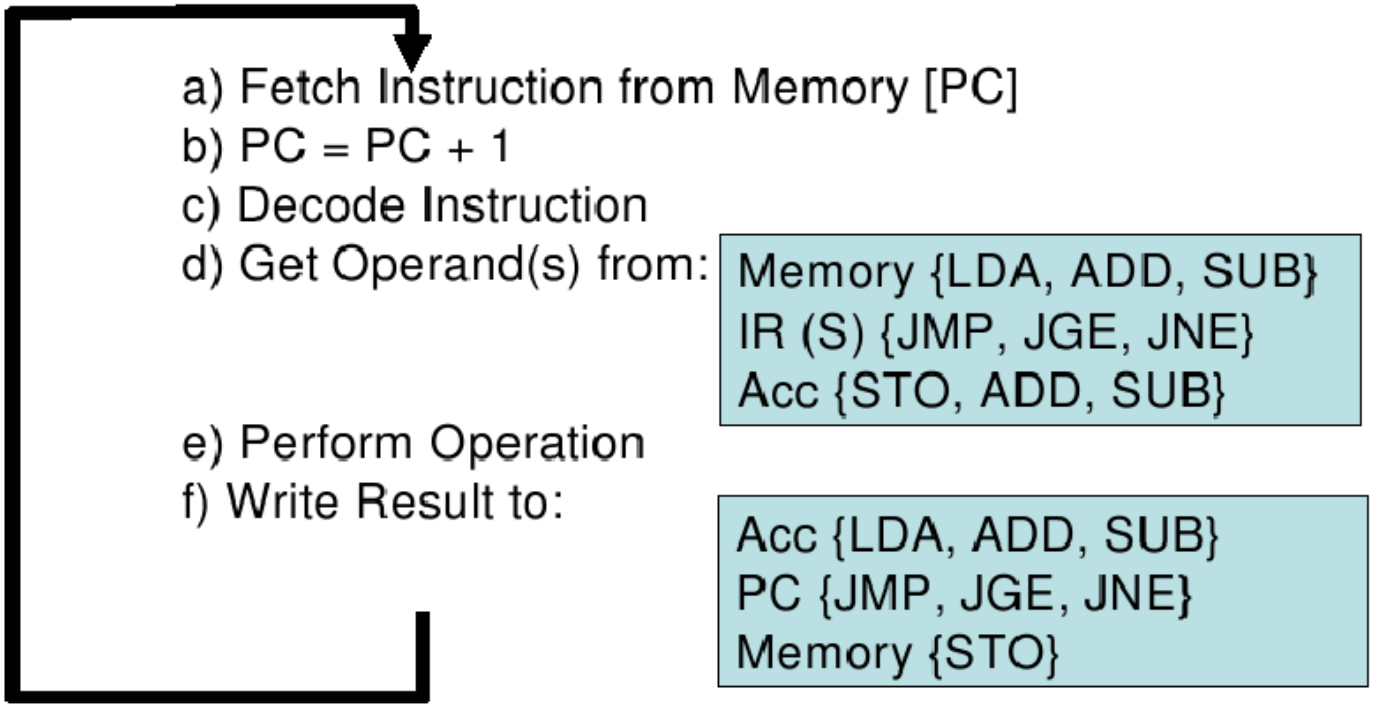
1) Access the memory operand and perform the desired operation
2) Fetch the next instruction to be executed

The processor must start in a known location – we can do this with a reset.

MU0 control logic

Instruction Execution Sequence

Like any CPU, MU0 goes through the three phases of execution:
These are repeated indefinitely. In more detail …

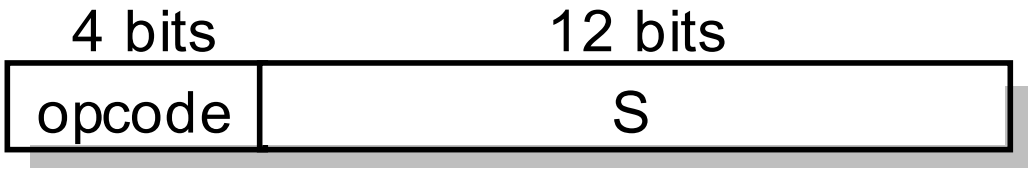


The MU0 instruction format

12 bit address space => 8k memory

4096 individually addressable memory locations

4 bit opcode => 16 possible assembly language
Commands only use 8! - good practice



The MU0 instruction set

First four have two memory accesses and will need two clock cycles

Last four could execute in one cycle

Instructi	Opcod	Effect
LDA S	0000	ACC := mem$_{16}$[S]
STO S	0001	mem$_{16}$[S] := ACC
ADD S	0010	ACC := ACC +
SUB S	0011	ACC := ACC -
JMP S	0100	PC := S
JGE S	0101	if ACC >= 0 PC := S
JNE S	0110	if ACC !=0 PC := S
STP	0111	stop

The MU0 control

Next we need to determine exactly what controls (logic levels) are needed to make the datapath execute the correct functions given the op-code

We assume that all registers change state on the rising edge of the clock (c.f negative edge Furber – probably because external SRAM used for the actually memory is posedge triggered??).

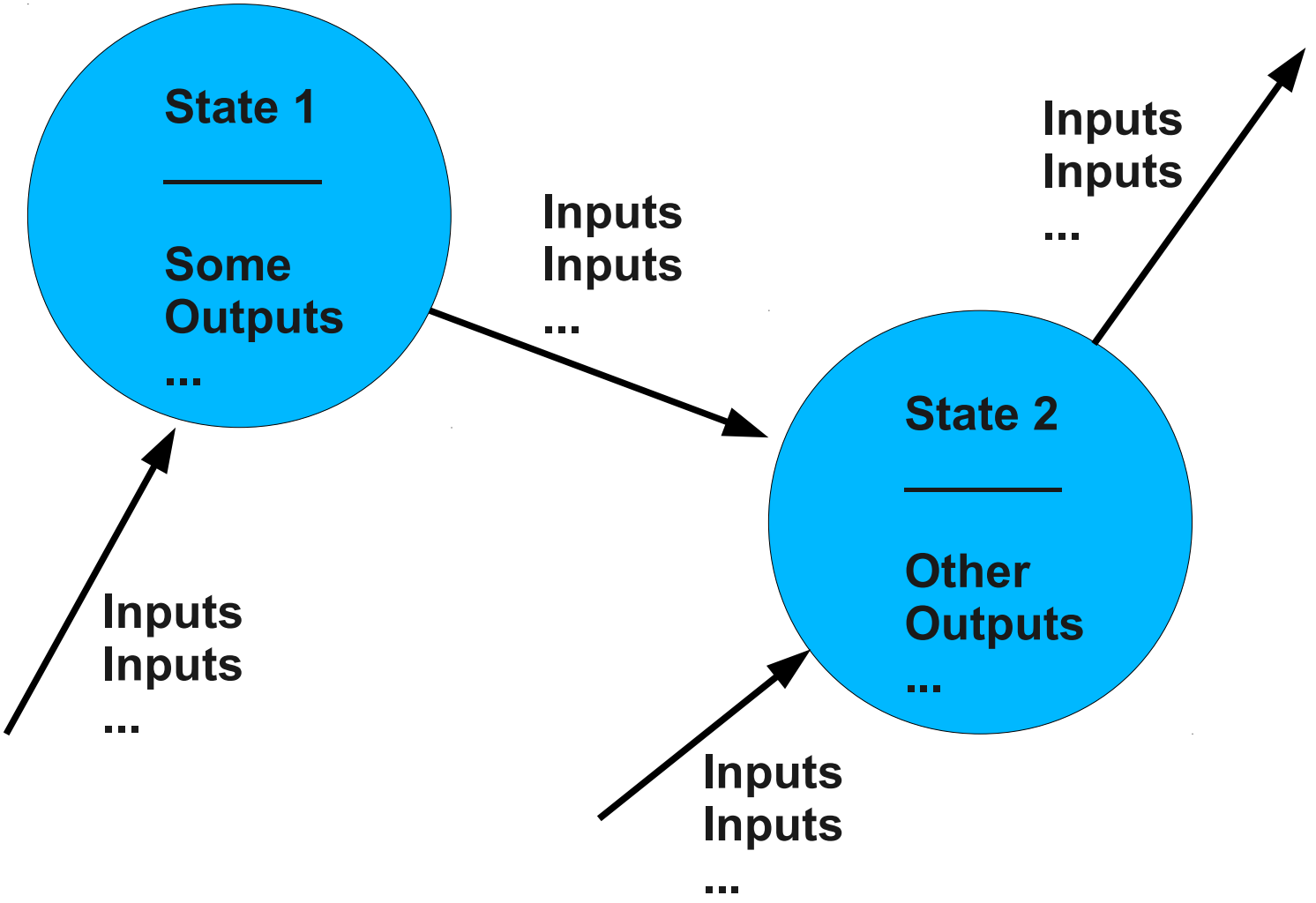
For the registers the control signals prevent or disallow transitions at the clock.

There are also feedback control signals from the datapath to the control FSM … opcode bits, signals from the **accumulator** indicating whether its contents are zero or negative which control the respective conditional jump instructions.

All we need to do is develop a two state FSM to generate the control signals

Since there are just two states and lots of control outputs => do a NS table and forget the SD approach

FSMs have no memory of outputs

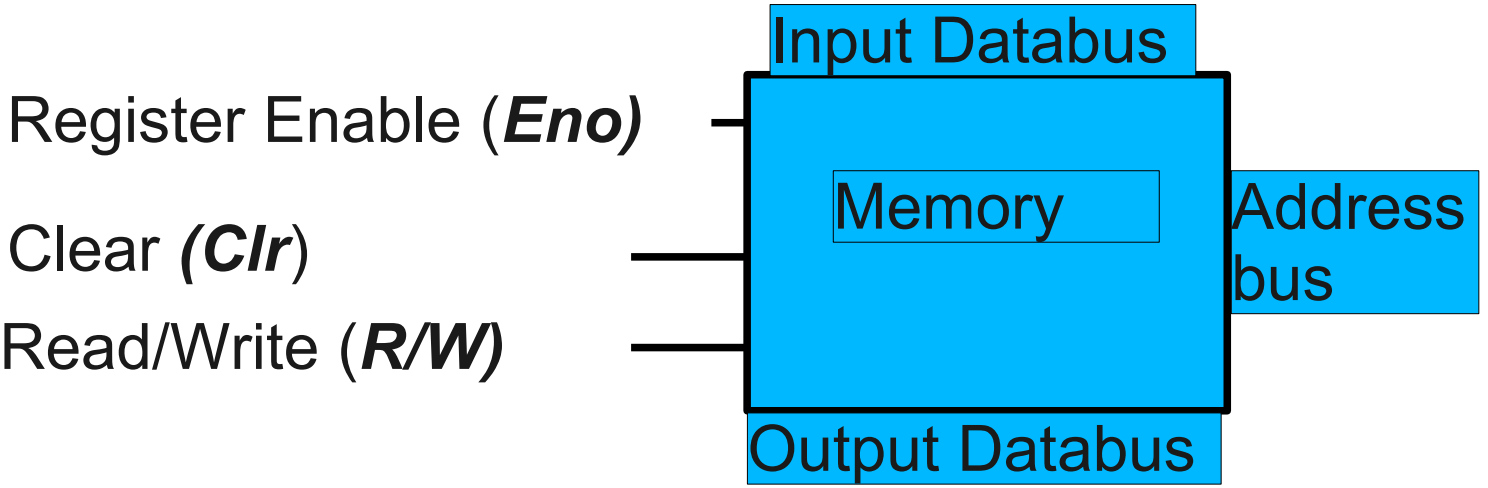


Memory comes from memory

Here is how – a 16 bit register...

```
module vreg16( clk, q, d, en, rs );

    output  [15:0] q;
    input   [15:0] d;
    input          clk;
    input          en;
    input          rs;

    reg     [15:0] q;

    always @(posedge clk) begin

        if(en && ~rs) q  <= d;
        else if(en &&  rs) q <=  16'h0;
        else q <= q;

    end

endmodule // v_reg16
```

Dealing with output “don’t cares”



From Furber

The “don’t cares” in FSM outputs set have a different meaning.

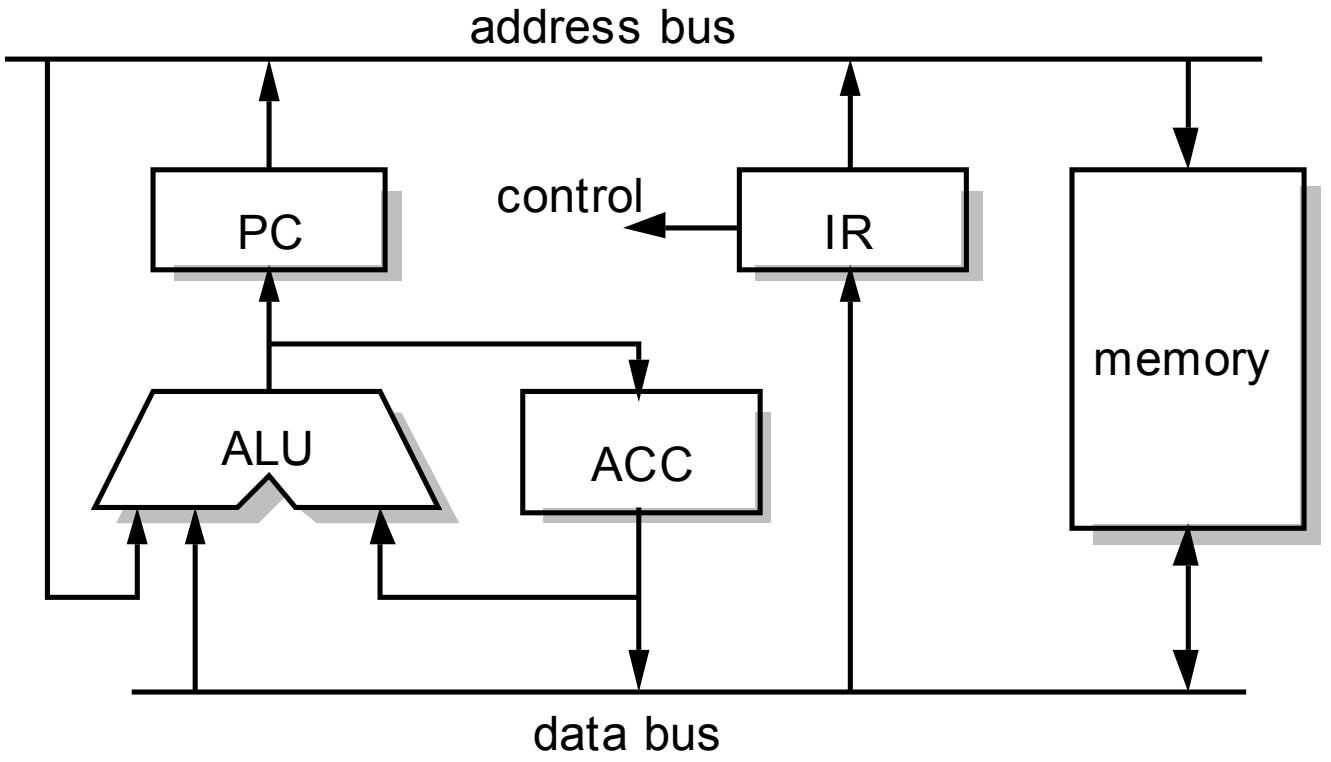
As the control state must generate these signals we must eventually define what they will be (Not X's)

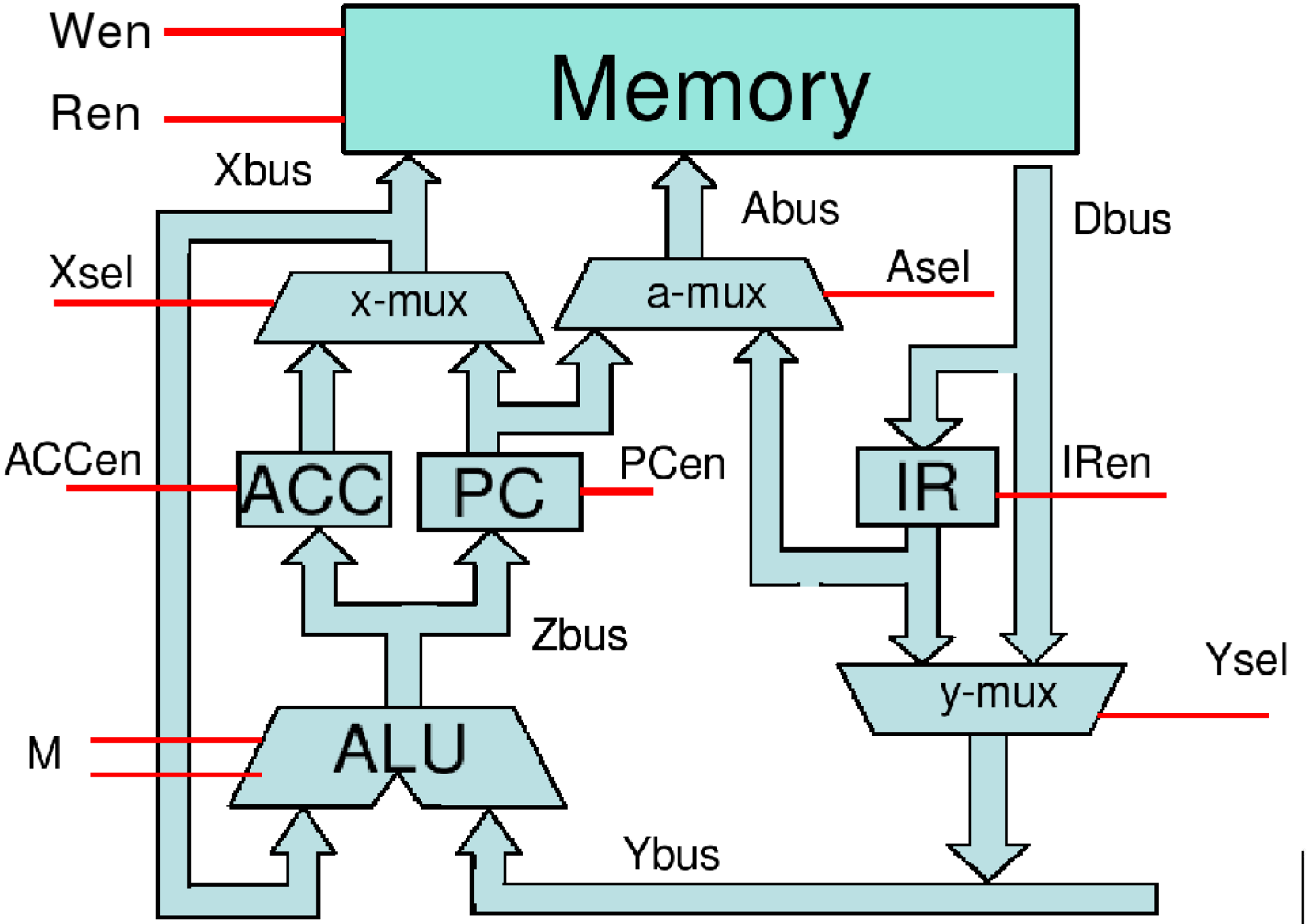
Note that ***Eno*** is always defined; this is because it is essential to know if the total register is to change or not on any given clock edge. If it is changing (***Eno*** = 1) then ***R/W*** and ***Clr*** control what it is doing; however if it is not changing (***Eno*** = 0) then it doesn’t matter what value is presented to the register – it will ignore it.

Allowing latitude at this time gives more freedom in the logic reduction, simpler equations and thus smaller (& faster) circuits.

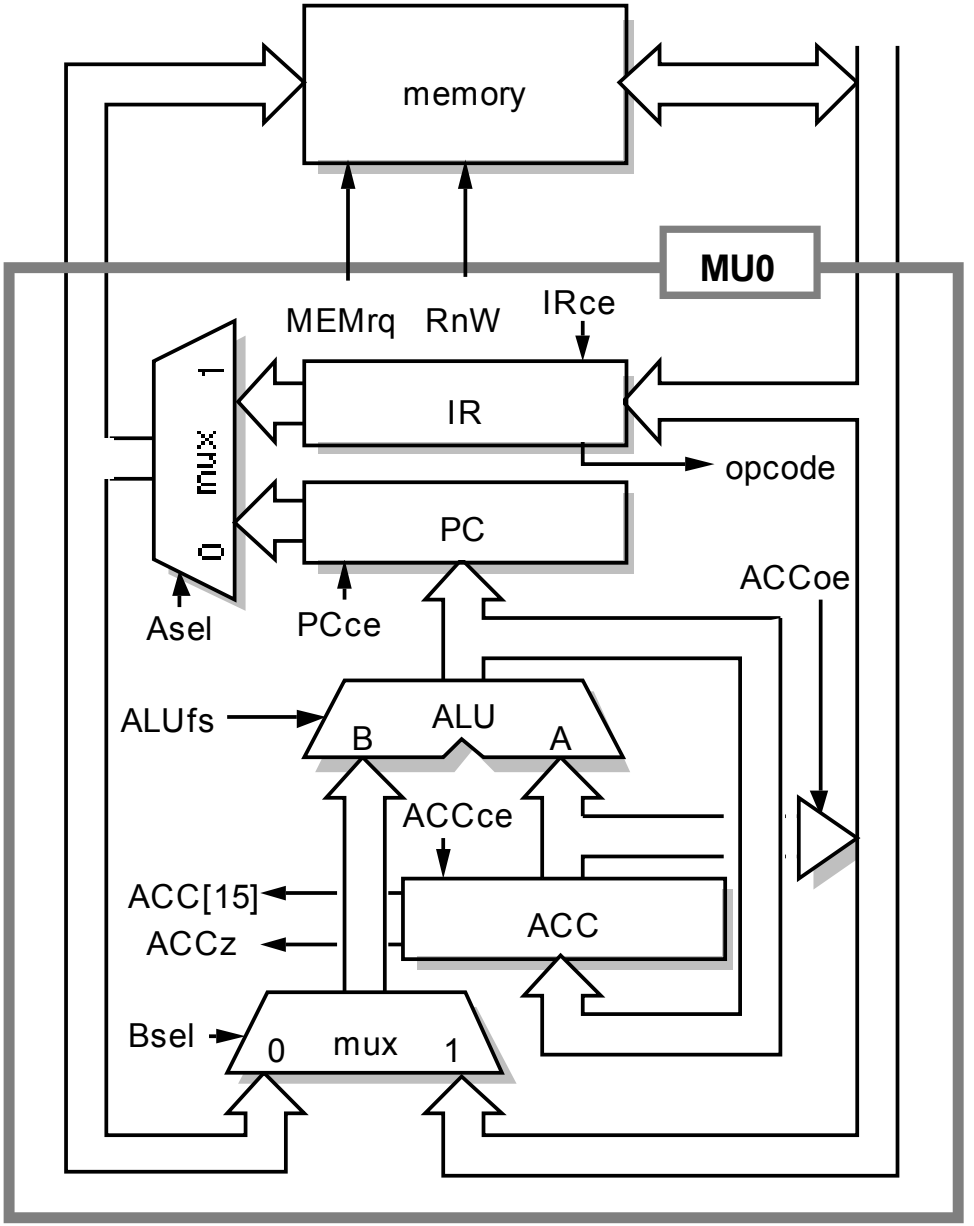
THE AUSTRALIAN NATIONAL UNIVERSITY

MU0 datapath example




Wen
Ren
Memory
Xbus
Abus
Dbus
Xsel
x-mux
a-mux
Asel
ACCen
ACC
PC
PCen
IR
IRen
Zbus
Ysel
y-mux
M
ALU
Ybus

MU0 register transfer level organization



MU0 control logic

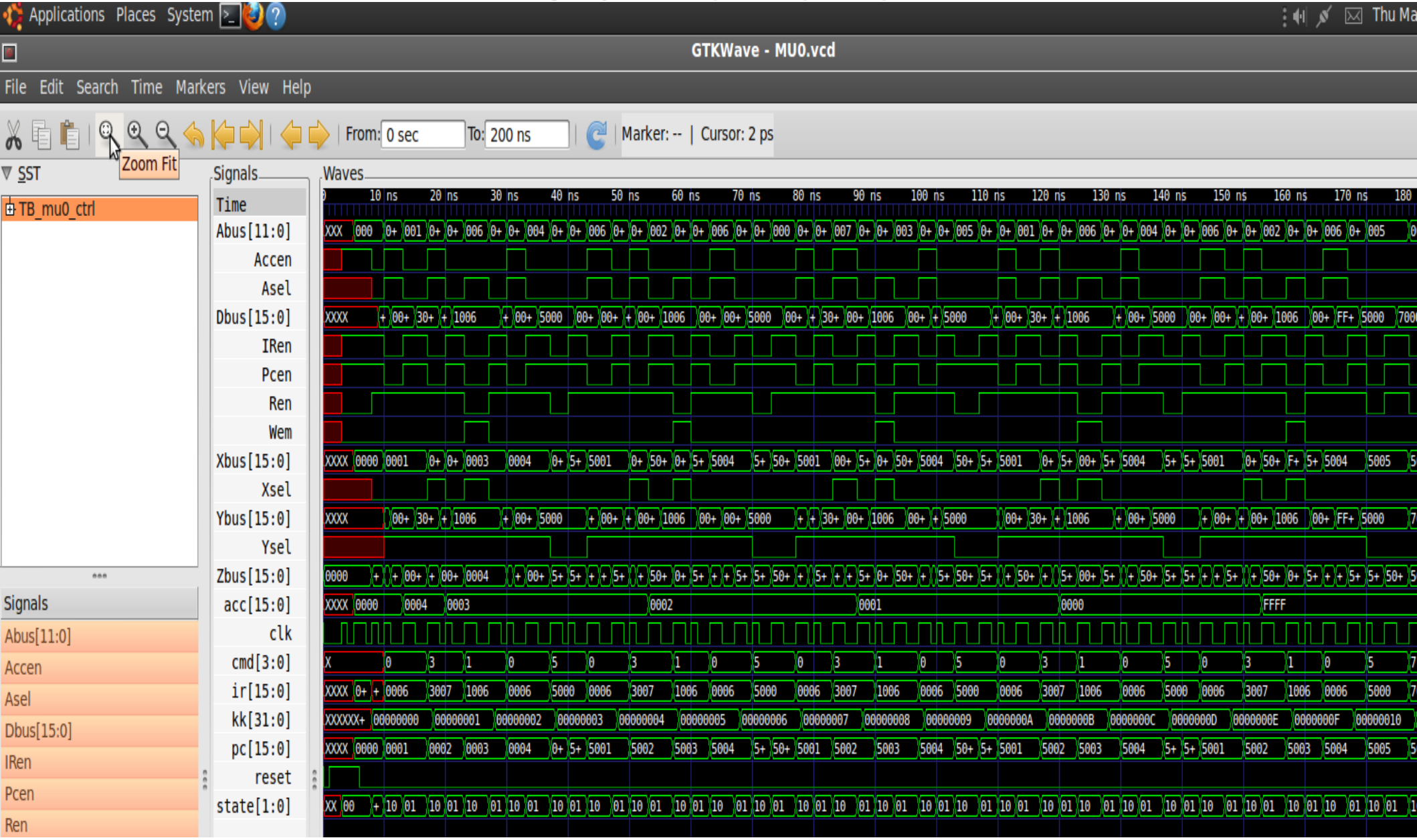
Inputs						Outputs									
	Opcode		Ex/ft	ACC15			Bsel		PCce		ACCoe		MEMrq	Ex/ft	
Instruction		Reset		ACCz		Asel	ACCce		IRce		ALUfs		RnW		
Reset	xxxx	1	x	x	x	0	0	1	1	1	0	=0	1	1	0
LDA S	0000	0	0	x	x	1	1	1	0	0	0	=B	1	1	1
	0000	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
STO S	0001	0	0	x	x	1	x	0	0	0	1	x	1	0	1
	0001	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
ADD S	0010	0	0	x	x	1	1	1	0	0	0	A+B	1	1	1
	0010	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
SUB S	0011	0	0	x	x	1	1	1	0	0	0	A-B	1	1	1
	0011	0	1	x	x	0	0	0	1	1	0	B+1	1	1	0
JMP S	0100	0	x	x	x	1	0	0	1	1	0	B+1	1	1	0
JGE S	0101	0	x	x	0	1	0	0	1	1	0	B+1	1	1	0
	0101	0	x	x	1	0	0	0	1	1	0	B+1	1	1	0
JNE S	0110	0	x	0	x	1	0	0	1	1	0	B+1	1	1	0
	0110	0	x	1	x	0	0	0	1	1	0	B+1	1	1	0
STOP	0111	0	x	x	x	1	x	0	0	0	0	x	0	1	0

MU0 machine language program

prog.lst

	Instruct	Opcod	Effect
0006	LDA S	0000	ACC := mem16[S]
3007	STO S	0001	mem16[S]:= ACC
1006	ADD S	0010	ACC := ACC +
0006	SUB S	0011	ACC := ACC -
5000	JMP S	0100	PC := S
7000	JGE S	0101	if ACC >= 0 PC :=
0004	JNE S	0110	if ACC !=0 PC := S
0001	STP	0111	stop

MU0 Function



MU0 – Extensions

- MU0 is a simple processor but not useful as a compiler target.
- Some extensions seem appropriate

  - ***Extending the address space***
  - ***Adding more addressing modes***
  - ***Allowing the PC to be saved in order to support a subroutine mechanism***
  - ***Adding more registers, supporting interrupts, etc...***
  - ***More peripherals – watchdog timer, ..***

. Overall MU0's instruction set is not a good place to start so let us redesign.

Where to start?

- Let us start with the core of the microprocessor functionality
  - The instruction

- ***Let us looking at a basic ADD for example***

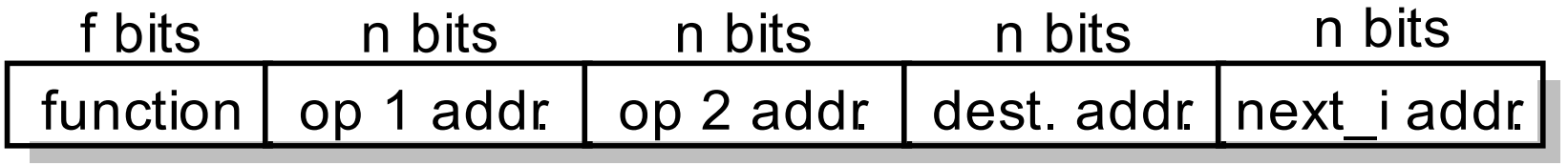
  - Some bits to differentiate from other instructions
  - Some bits to specify operand addresses
  - Some bits to specify where the results should be placed **- desitnation**
  - Some bits to specify the address of the next instruction

A 4-address instruction format

→ ***Assembly language instruction format might look like***

ADD d, s1, s2, next_i ; d := s1 + s2

Requires (4n + f bits)

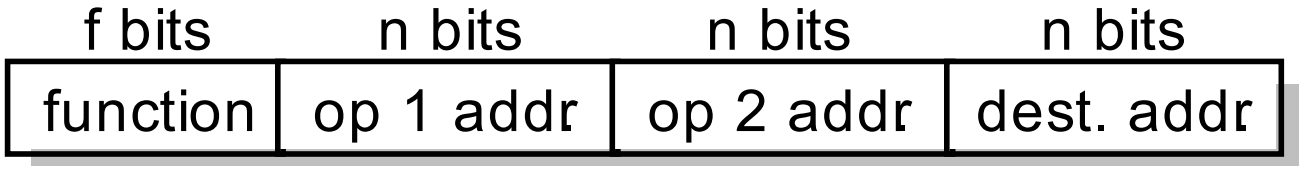


A 3-address instruction format

One way to reduce the number of bits required for each instruction is to make the address of the next instruction implicit.

We assume that the next instruction is PC+ Sizeof(instruction), (note that in MU0 the default next instruction was at PC+1. But if We generalise then maybe there can be more than one address to contain the contents of an instruction)

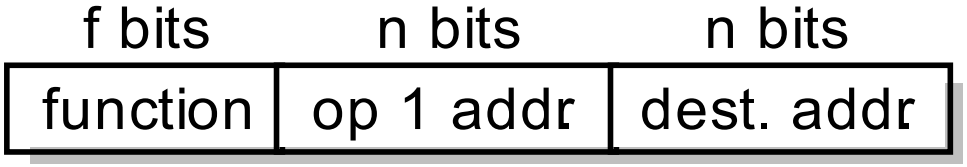
ADD d, s1, s2; d := s1 + s2



A 2-address instruction format

A further saving can be made by making the destination register the same as one of the source registers

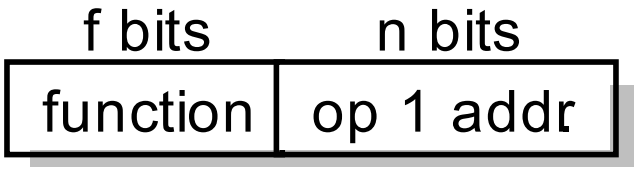
ADD d, s1; d := d + s2



A 1-address (accumulator) instruction format

If the destination register is implicit then it is often Called the **accumulator**

ADD s1; accumulator := accumulator + s2



A 0-address instruction format

Finally all registers may be made implicit by introducing An evaluation stack

ADD ; top_of_stack := top_of_stack + next_on_stack

Examples of n-address use

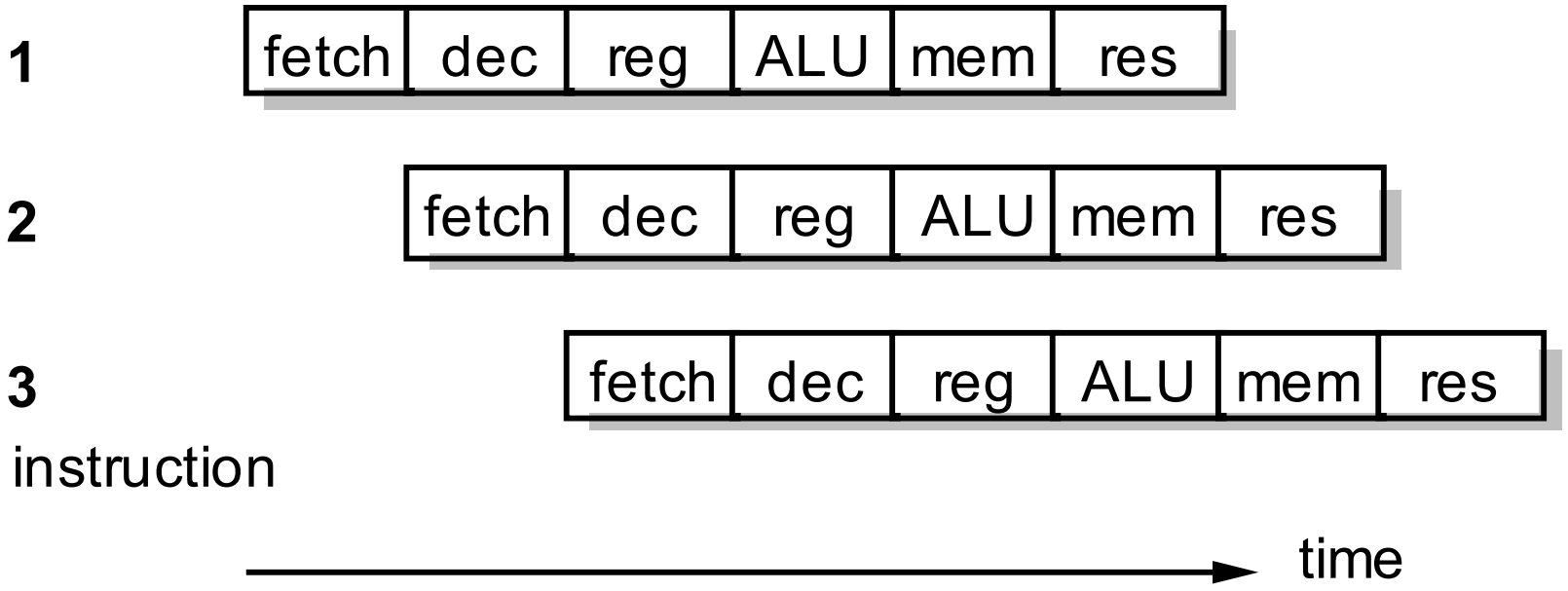
All of the above have been used in processor instruction sets apart from the 4-address form which, although it is used internally in some ***microcode(??)*** designs is **unnecessarily expensive**

- **The Inmos transputer uses a 0-address evaluation stack architecture**
- **The MU0 example in the previous section is a 1-address architecture**
- **The Thumb instruction set used for high code density in the ARM micros is predominatly of the 2-address form**
- **The standard ARM instruction set uses a 3-address architecture**

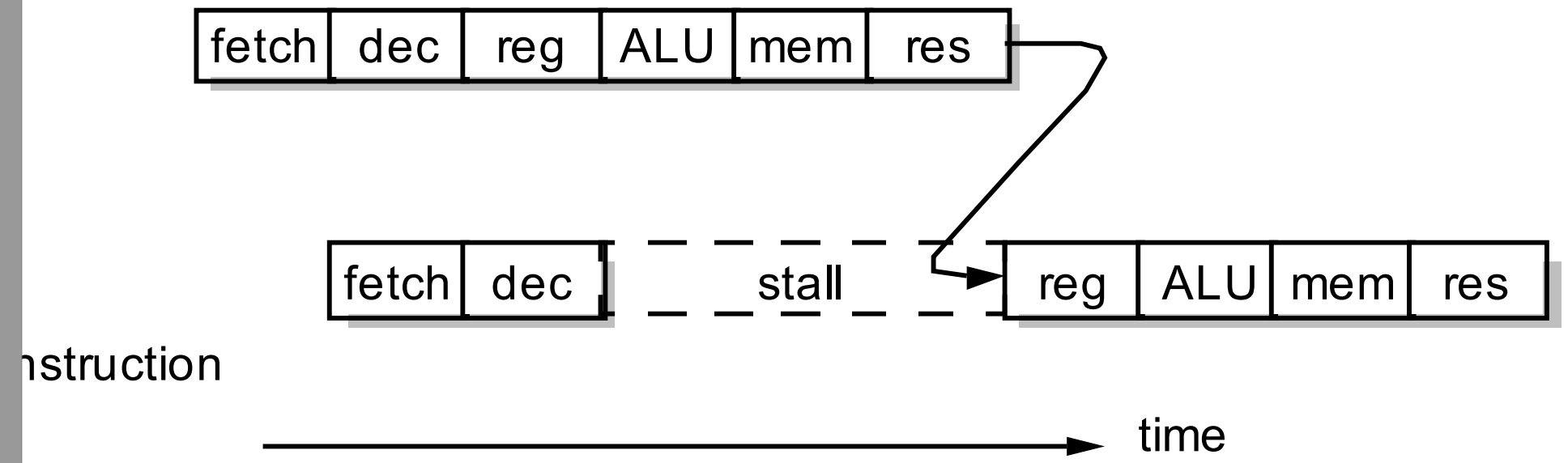
Typical dynamic instruction usage

Instruction type	Dynamic usage
Data movement	43%
Control flow	23%
Arithmetic operations	15%
Comparisons	13%
Logical operations	5%
Other	1%

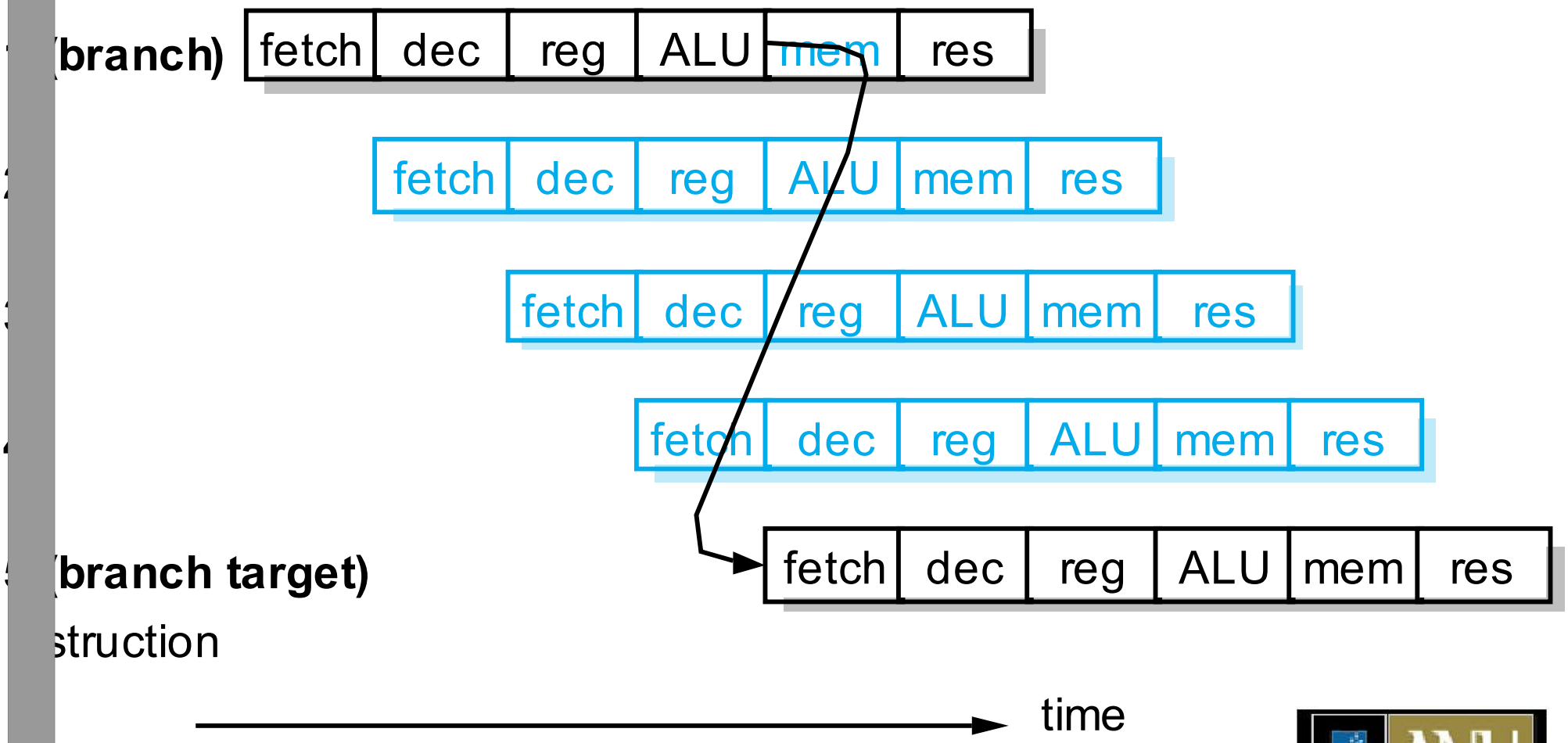
Pipelined instruction execution



Read-after-write pipeline hazard



Pipelined branch behaviour



Naming ARM

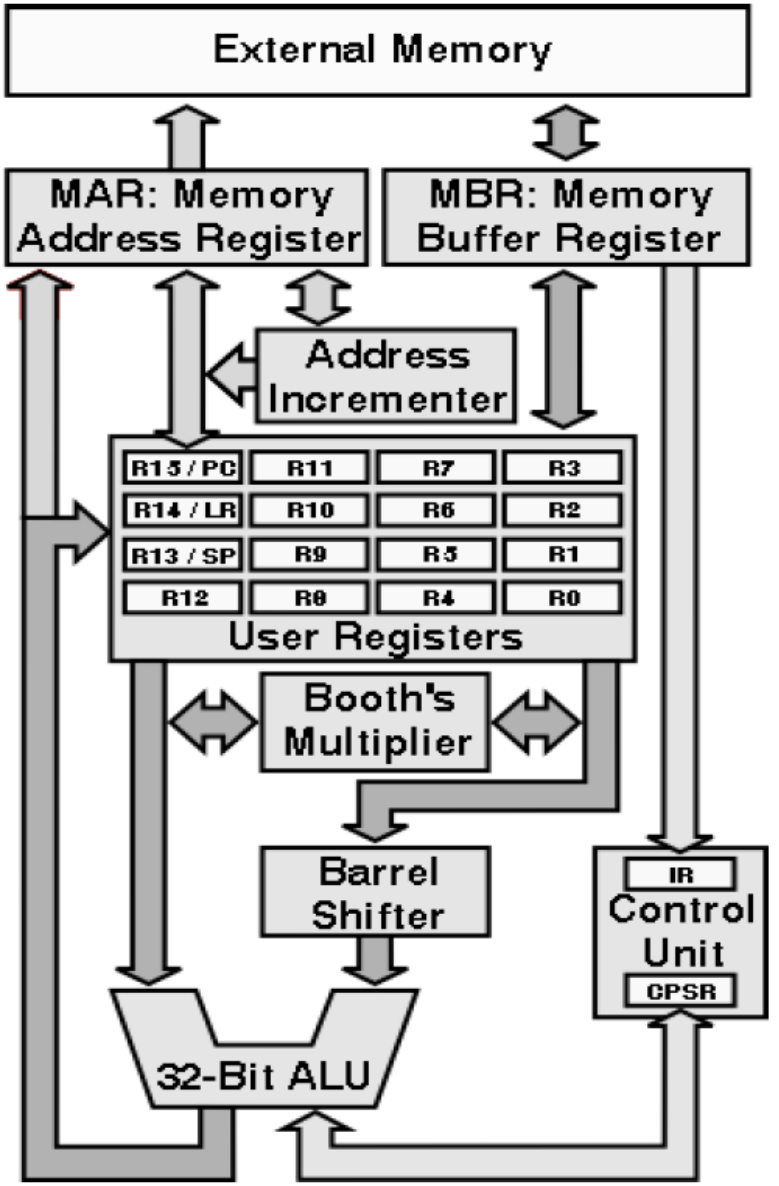
- ARMxyzTDMIEJFS
  - x: series
  - y: MMU
  - z: cache
  - T: Thumb
  - D: debugger
  - M: Multiplier
  - I: Interrupt
  - E: Enhanced
  - J: Jazelle
  - F: Floating-point
  - S: Source

Popular ARM architecture

- ARM7TDMI
  - 3 pipeline stages
  - One of the most used ARM-version (for low-end systems)
- ARM9TDMI
  - Compatible with ARM7
  - 5 pipeline stages
  - Separate instruction and data cache
- ARM11

ARM architecture

- Load/store architecture
- A large array of uniform registers
- Fixed-length 32-bit instructions
- 3-address instructions

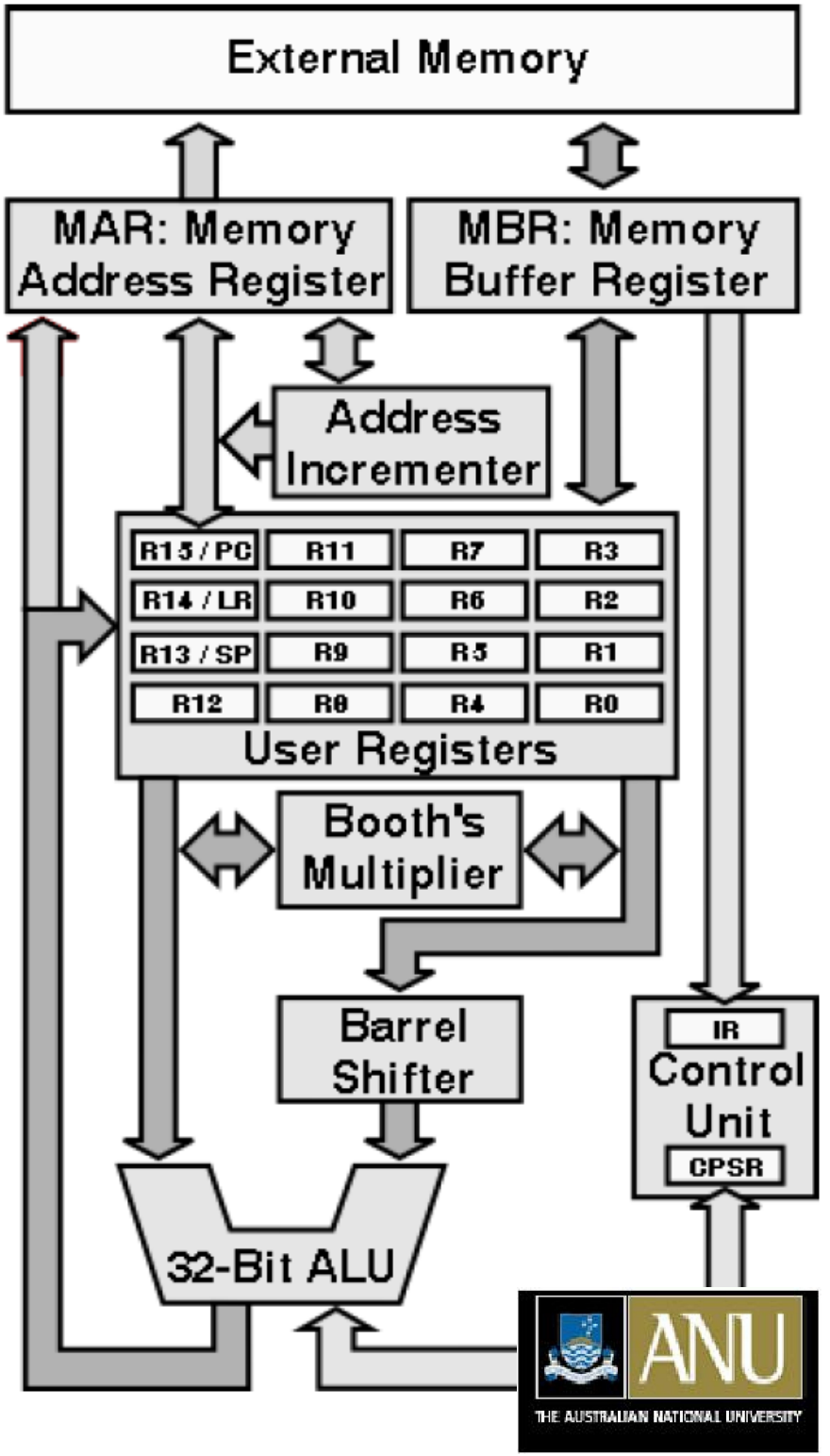


Processor modes

Processor mode		Description
User	usr	Normal program execution mode
FIQ	fiq	Supports a high-speed data transfer or channel process
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	A protected mode for the operating system
Abort	abt	Implements virtual memory and/or memory protection
Undefined	und	Supports software emulation of hardware coprocessors
System	sys	Runs privileged operating system tasks

ARM architectu

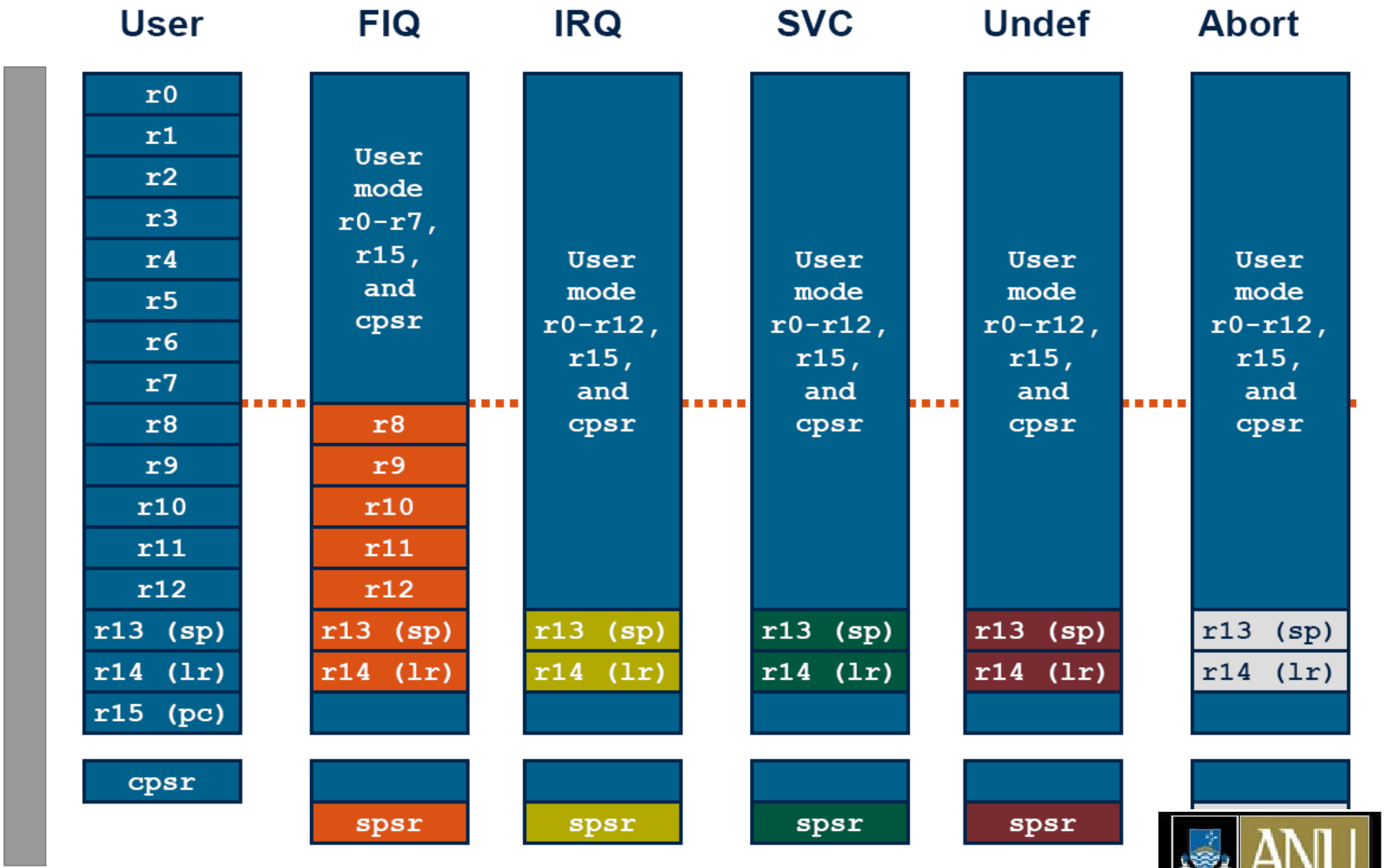
- 37 registers
  - 1 Program counter
  - 1 current program status registers
  - 5 saved program status registers
  - 30 general purpose registers



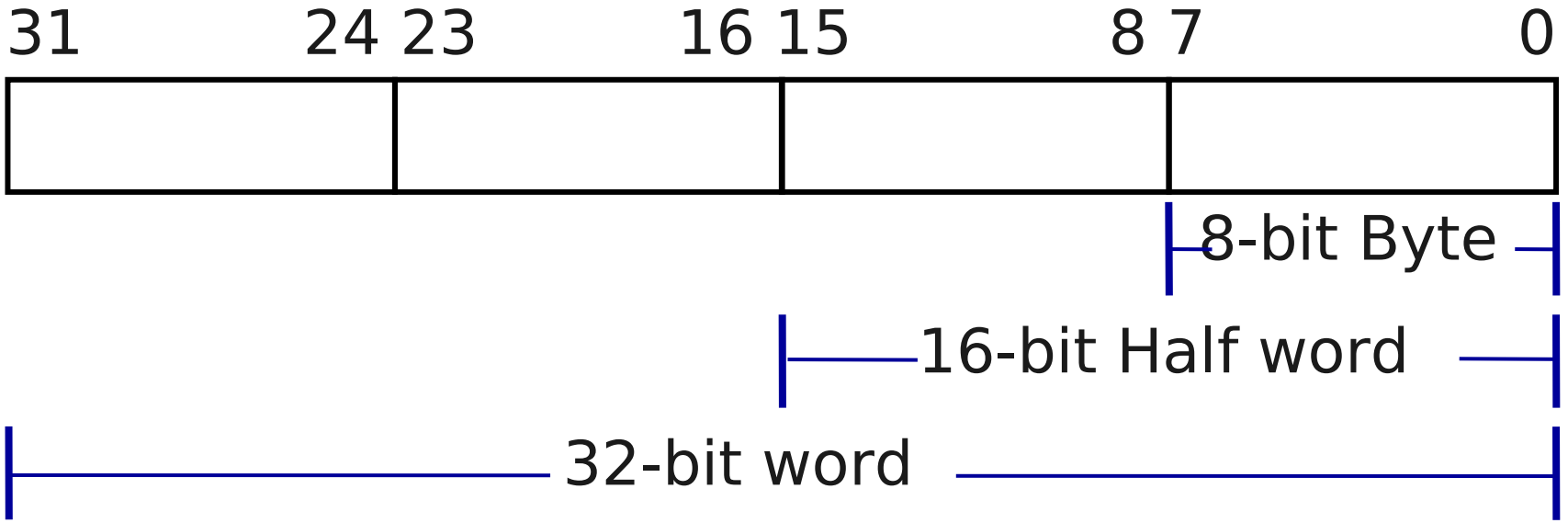
Registers

- Only 16 registers are visible to a specific mode. A mode could access
  - A particular set of r0-r12
  - r13 (sp, stack pointer)
  - r14 (lr, link register)
  - r15 (pc, program counter)
  - Current program status register (cpsr)

Register organization



General-purpose registers

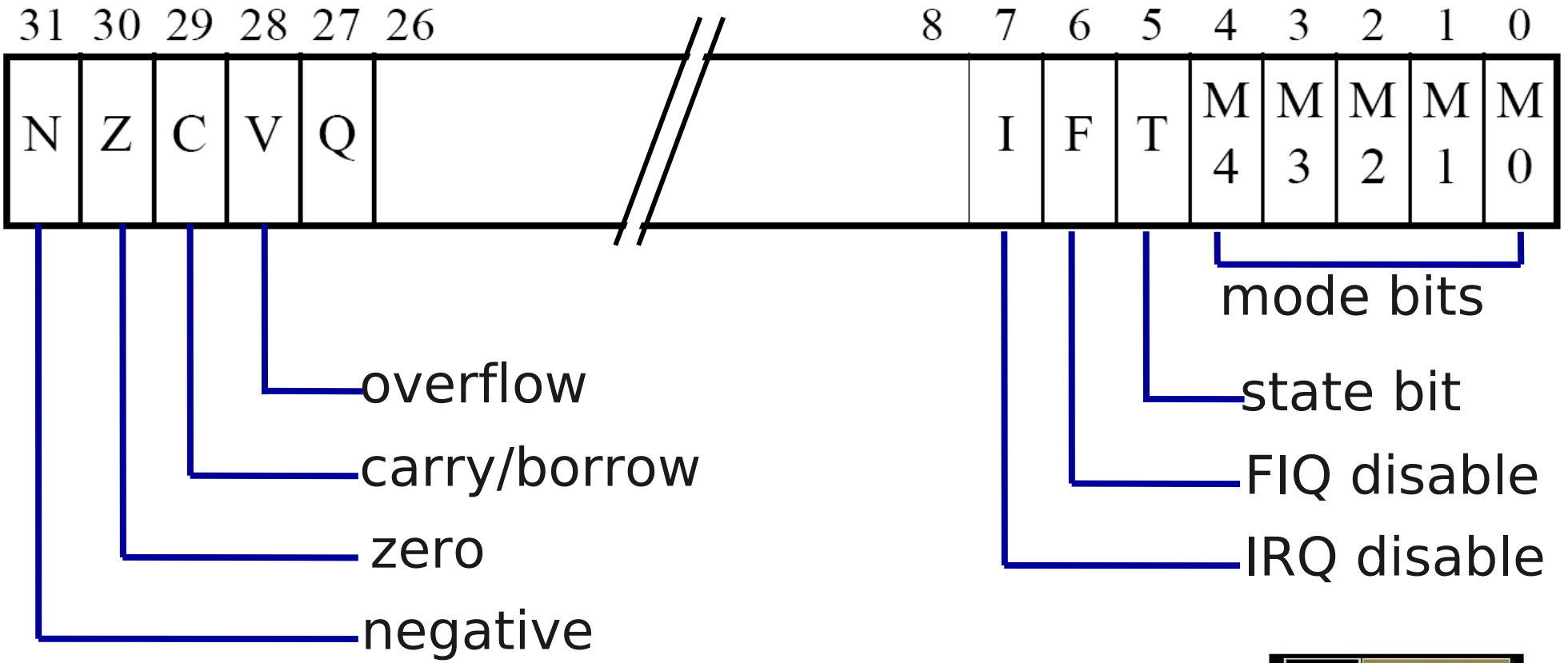


- 6 data types (signed/unsigned)
- All ARM operations are 32-bit. Shorter data types are only supported by data transfer operations.

Program counter

- Store the address of the instruction to be executed
- All instructions are 32-bit wide and word-aligned
- Thus, the last two bits of pc are undefined.

Program status register (CPSR)



Summary

- Load/store architecture
- Most instructions are RISCy, operate in single cycle.
  - Some multi-register operations take longer.
- All instructions can be executed conditionally.

SEARCH

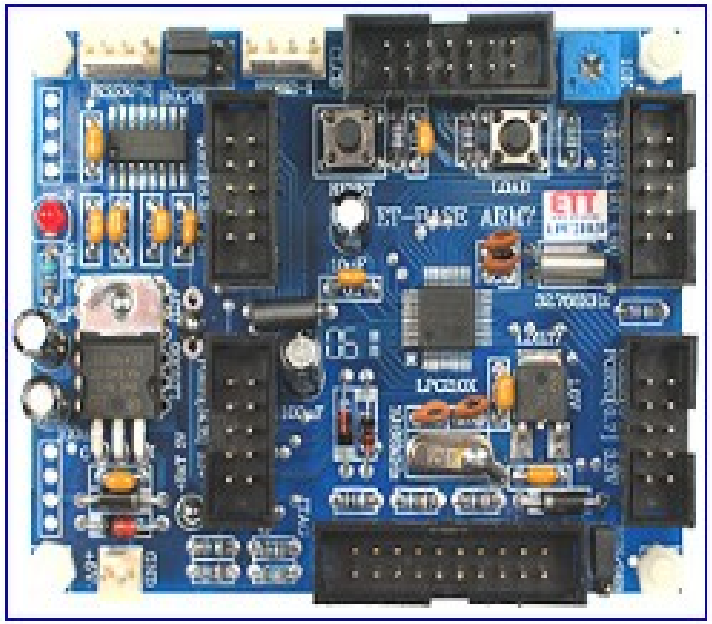
der Status

ents
mentBoards
Boards
rs
C832 Controller
C842 Controller
2103
2368
7024
ega
ega8535
ny2313
C51AC3
C51ED2
LP4052
2
RM Stamp
TM32 Stamp

Welcome to Futurlec. The ELECTRONIC COMPONENTS Superstore. To find the component your looking for, either search by Part Number or visit the relative department.

Need Help.

CONTROLLERS



ARM2103 Controller - Technical Data

- **Microcontroller:** LPC2103
- **Main Crystal:** 19.66MHz
- **Speed:** Up to 59.9842MHz
- **Processor Language:** ARM

Memory

- **Program Flash Memory (Internal):** 32kBytes
- **RAM Memory - Scratchpad (Internal):** 8 kBytes
- **EEPROM Memory (Internal):** None

Input/Ouput

- **I/O Points Available:** 32
- **I/O Points Connection:** IDCC Connector

Linux usage on Intel (4 level Von Neuman)

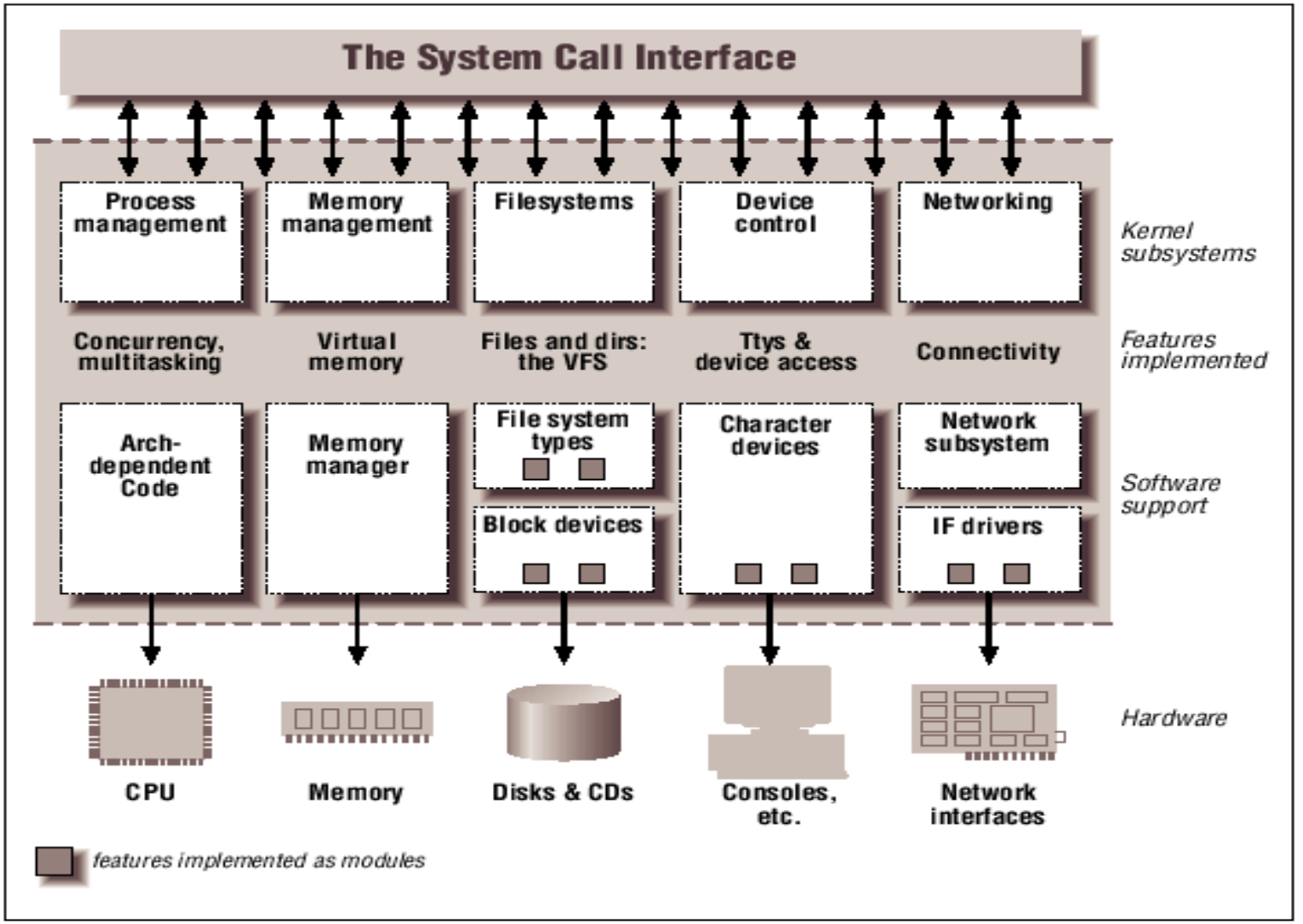


Figure 1-1. A split view of the kernel