



MODULAR PROGRAMME
ASSESSED COURSE-WORK SPECIFICATION

Module Details:

Module Code UFMEVP-20-2	Run 11SEP/1AY	Module Title Architecture of CPUs with VHDL
Module Leader Nigel Gunton		Module Tutors Nigel Gunton
Component and Element Number B2		Weighting:(% of the Modules Assessment) 56.25%
Element Description Course-work		Total Assignment time 18 hours

Dates:

Date issued to students: 20th January 2012	Date to be returned to Students
Submission Place: Project Room, 2Q30 open 09.00-18.00	Submission Date: 29th March 2012
	Submission Time: 14.00

Deliverables:

As listed on the Assignment spec sheet

Module Leader Signature:

--

You are strongly recommended to work in pairs for the CPU design.

0. Overview

Modern systems on silicon often require a complex control unit or sequencer. This can have similarities to the control unit of a CPU in that it has to generate a sequence of control signals dependent on the current input and current instruction. It is also possible that a design for a digital system is best implemented by a simple decoder without requiring the complexity of a CPU core, or incurring the cost of devices with CPU cores already embedded in them.

To better understand this design process, you are required to develop and test a simple microprocessor. Although you can choose to develop your own design from scratch, it is **STRONGLY** recommended that your design is based on the 'Carpinelli RS-CPU'¹. Possible alternative microprocessor designs are listed on the module web page.

The marking scheme indicated below is based on the assumption that you will follow the design for Carpinelli's RS_CPU. If you choose to implement a different processor then the marks allocation will be altered to reflect the development effort in the first stages. You should still follow the sequence of development indicated below.

The design process will be supported by the lectures and support will be given in the lab sessions. However the design process summarized here uses Carpinelli's relatively simple CPU as a model.

1. Guidelines & Milestones:

1.1. Marks Allocation

Marks will be broadly allocated along the following lines²:

- a) Implementation of the RS-CPU using either a hardwired or VHDL state machine control unit to include registers and ALU, all verified, will be marked to a maximum of 60%. The remaining 40% will awarded to extensions of the basic design. Marks will be awarded for design and for implementation. The balance between design and implementation marks will depend on the complexity of the extensions.
- b) Implementation of the RS-CPU using a micro-sequencer, including registers and ALU, all verified, will be marked to a maximum of 75%. The remaining 25% will be for extensions to the design, for example micro-sequencer subroutines or as outlined below. These marks will be distributed between design and implementation.
- c) Design and implementation of an original CPU design will be marked to a maximum of 70% for the initial design and implementation of the control unit for your design including verification. Up to the remaining 30% will be awarded for the full integration and verification of your design.
- d) Implementation and verification of an alternative design, such as the JRISC, will be marked in the same manner as (c).

1.2. Possible Extensions

Status Register

Extend the status register to include Carry, Overflow, and Negative flags plus additional instructions to make use of these registers.

Multiplier Unit

Implement a 'shift-add' multiplier and verify it. This will require additional control signals and instructions.

Stack Pointer

Implementation of a stack pointer and additional instructions to provide 'branch and return'.

¹ From John Carpinelli's design for a relatively simple CPU

² Note that in all cases a proportion of the marks will be for the quality and standard of the submitted documentation.

This can be extended further to provide interrupt handling.

Additional addressing modes

Extend the addressing modes to include, for example, immediate and indexed addressing.

Implement the RS-CPU in an FPGA

1.3. Laboratory Milestones

16/02/12

For option (a) you should aim to have the control unit completed and at least partially verified by 16th February. If you are working in pairs then you should be developing the ALU in parallel with a similar target for verification and sign-off of the ALU.

23/02/12

For option (a) registers should be completed and verified.

For options (b) and (d) you should aim to have the control unit mainly functioning by the 23rd February even if not fully verified.

Option (c) should have the concrete RTL completed along with the final register section and a clear decision for the control unit implementation.

1/03/12

(a) Full integration of registers, ALU and control unit. Integration testing and verification underway.

(b),(d) Integration and verification of control unit components completed, verification of registers in progress

(c) control unit ready for verification (hard-wired or FSM), in progress (micro-sequencer)

8/03/12

(a) design of extensions under way...or porting to Altera Quartus for FPGA implementation

(b),(d) integration testing and verification of complete CPU

(c) commencement of CPU integration and verification

15/03/12

(a) extension design complete, implementation commenced

(b),(d) integration and verification completed, design of extensions commenced

(c) verification completed, design review commenced

22/03/12

I will be away for this week

(a), (b), (c), (d) complete all documentation in preparation for submission on 29th March

29/03/12

SUBMISSION!!!! Laboratory demonstrations commence.

2. Verification

The verification of a design will normally take about 80% of the total development time. You should consider exactly what you are verifying. Implement a verification plan... what values will the inputs to the system be? For how long? What are the expected outputs? Consider the 'corner cases'. The Alliance tools provide a set of C functions for writing large sets of test inputs. An example of a verification plan and the associated pattern file is provided on the module web-page, see:

http://www.cems.uwe.ac.uk/~ngunton/vhdl/example_waveform_for_worksheet.pdf

3. Integration

An example structural model showing the integration of an RS-CPU, ROM and RAM is also provided on the module web-page. The ROM and RAM files are given. The ROM includes a sample programme for execution on the RS-CPU. The principle of integrating the registers, control unit and ALU is identical to that shown in the example.

4. Marking

Marks will be award in line with section 1.1. Within these guidelines,

- 25% of the marks will be from a viva/demonstration/discussion. Designs do not need to be fully or correctly working.

- 65% will be for the documentation of design and verification. You should provide top level VHDL for all unique components and integration, verification plans along with sample results and a table for all verification and integration showing 'pass/fail' for each test. Discussion of design decisions for extensions and novel designs.
- 10% for document quality.

5. Appendix 1

RS-CPU Specification

The design for the RS-CPU is by John D Carpinelli and can be found in his book *Computer Systems, Organization & Architecture* published by Addison Wesley, Boston, USA

The following diagrams and tables provide the design details of the CPU and the first stages of the development process for the hardwired control unit. The ALU design and its control signals are also provided.

This CPU can address 64 Kbytes of memory, each word is 8 bits wide. This is via address pins A[15..0] and bidirectional data pins D[7..0]. There are three programmer accessible registers :

AC The accumulator, AC, is used to receive the results of all arithmetic and logical operations that require 2 operands. It also provides one of the operands for these operations. It is also used for all data transfers between the CPU and memory.

R Register R is an 8-bit general purpose register that provides the second operand for all 2 operand arithmetic and logical operations. It can also be used for the temporary storage of data.

Z A one-bit zero flag, Z, is modified whenever an arithmetic or logical operation is performed.

The CPU also contains the following registers

AR 16-bit address register, AR, supplies an address to memory via A[15..0]

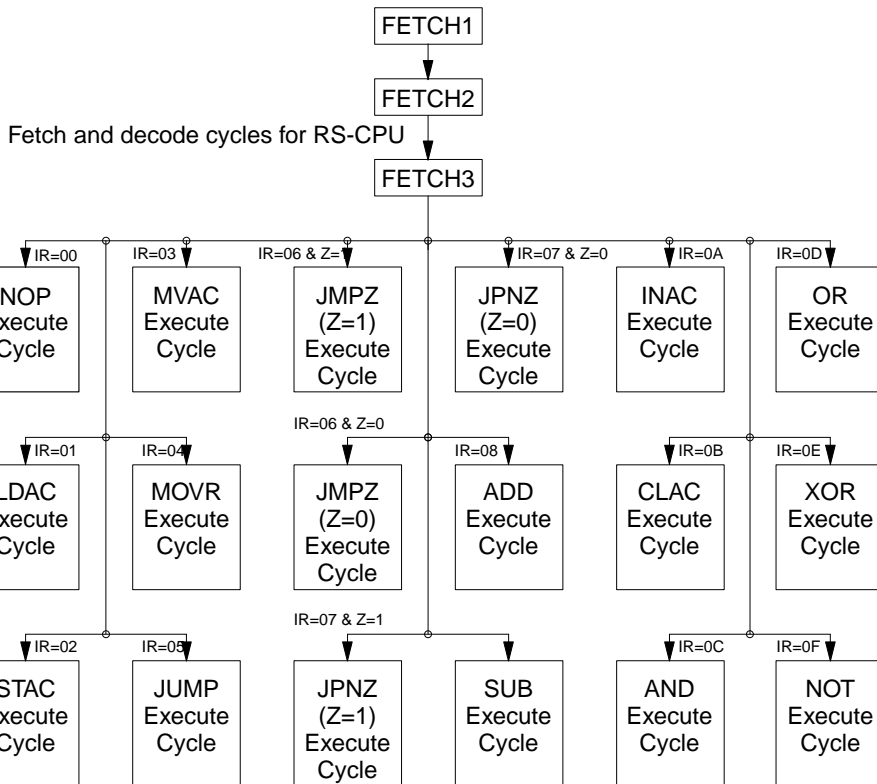
PC 16-bit program counter, PC, contains the address of the next instruction to be executed, or the address of the next required operand of the instruction.

DR 8-bit data register, DR, receives instructions and data from memory and transfers data to memory via D[7..0].

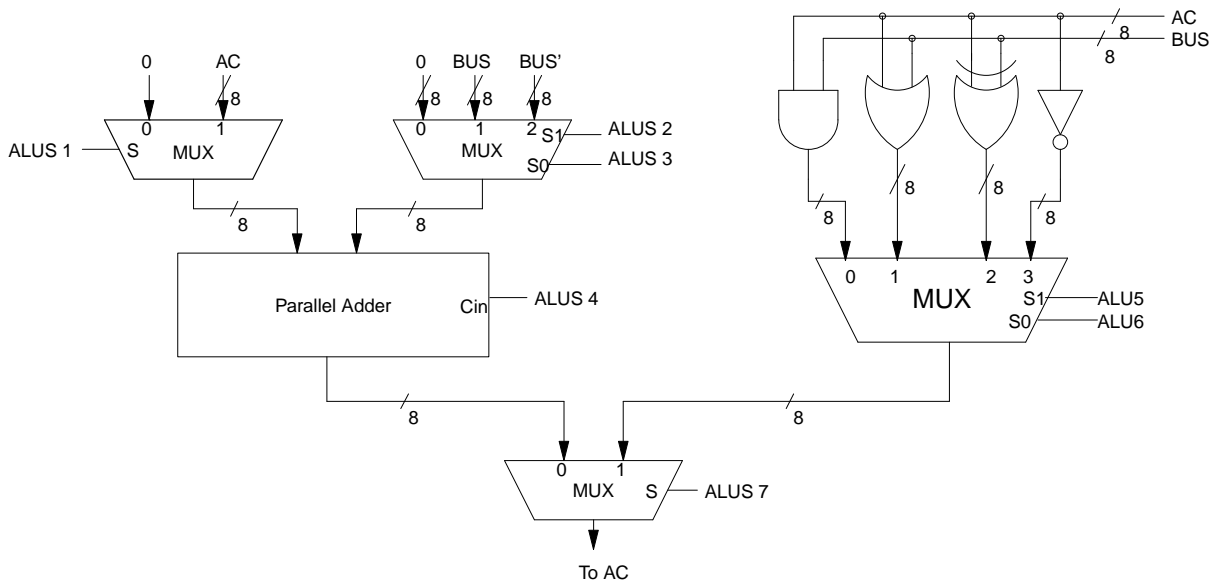
TR 8-bit temporary register, TR, stores data during instruction execution, eg. to hold the low byte of operand addresses.

Note that the program counter can hold either the address of an immediate operand or the address of the next instruction. In fact it is not known whether the address is that of an instruction or an operand until the current instruction is decoded.

All arithmetic and logic is done on the accumulator and register R for 2 operand operations, and the accumulator only for single operand operations, so no operands are required for this class of instruction.



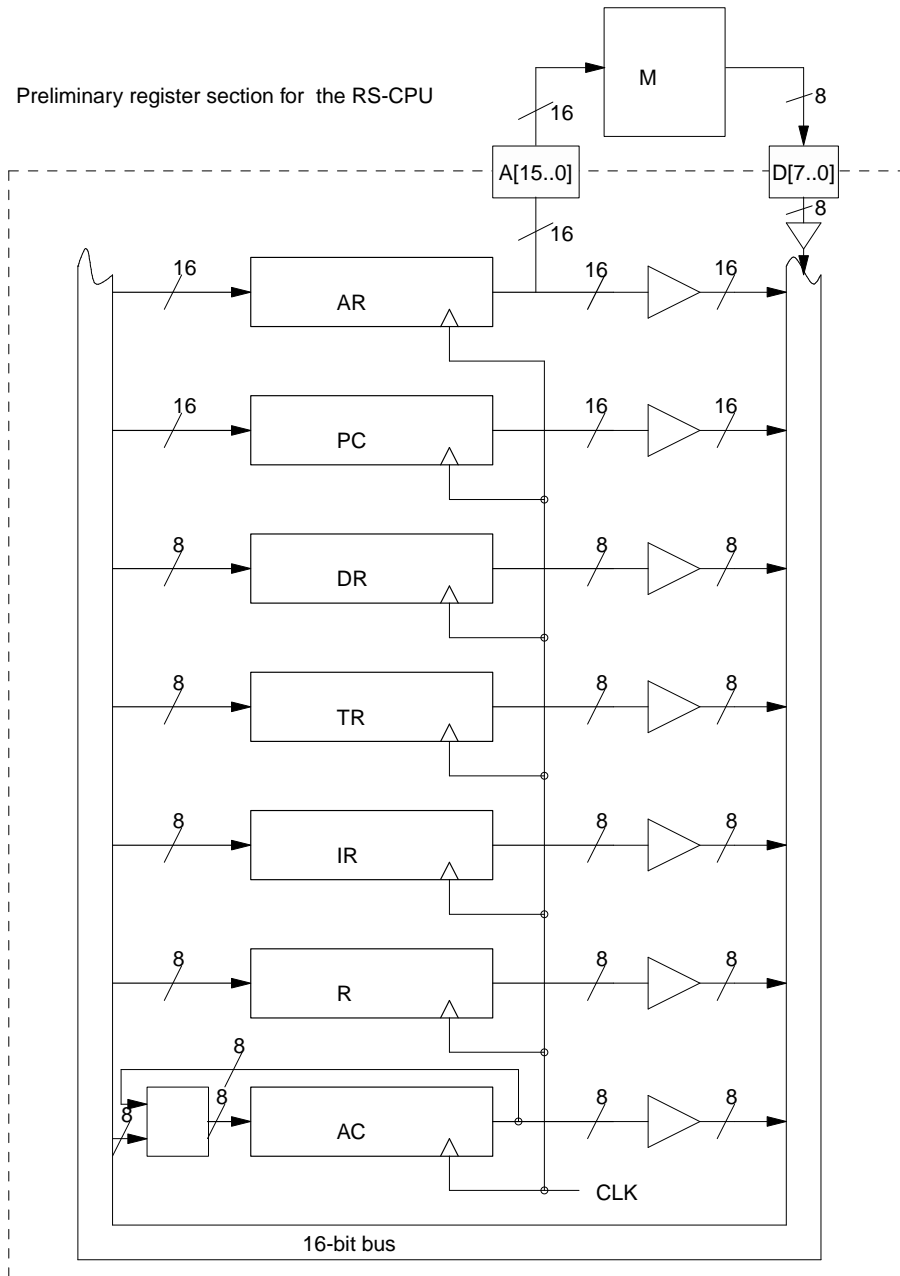
ALU implementation for the RS-CPU



Instruction set for a relatively simple CPU

Instruction	Instruction Code	Operation
NOP	0000 0000	No operation
LDAC	0000 0001 Γ	$AC \leftarrow M[\Gamma]$
STAC	0000 0010 Γ	$M[\Gamma] \leftarrow AC$
MVAC	0000 0011	$R \leftarrow AC$
MOVR	0000 0100	$AC \leftarrow R$
JUMP	00000101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF (Z=1) THEN GOTO Γ
JPNZ	0000 0111 Γ	IF (Z=0) THEN GOTO Γ
ADD	0000 1000	$AC \leftarrow AC + R$, IF (AC + R = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
SUB	0000 1001	$AC \leftarrow AC - R$, IF (AC - R = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
INAC	0000 0101	$AC \leftarrow AC + 1$, IF (AC + 1 = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
CLAC	0000 1011	$AC \leftarrow 0$, Z \leftarrow 1
AND	0000 1100	$AC \leftarrow AC \wedge R$, IF (AC \wedge R = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
OR	00001101	$AC \leftarrow AC \vee R$, IF (AC \vee R = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
XOR	0000 1110	$AC \leftarrow AC \oplus R$, IF (AC \oplus R = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0
NOT	0000 1111	$AC \leftarrow AC'$, IF (AC' = 0) THEN Z \leftarrow 1 ELSE Z \leftarrow 0

Preliminary register section for the RS-CPU



6. Appendix 2

Program specification & memory map (8 bit data, 16 bit Address)

- The program MUST reside in ROM and WILL be relocated to RAM after booting.
- The program is to read an 8 bit wide value in from an input port.
- The value is used as an index into a look-up table (LUT) held in ROM.
- The data (also 8 bits wide) at the indexed address is to be written out to an output port.

Memory map:

```
0x8080 DATA_IN
0x8000 DATA_OUT
0x07FF RAM_END
0x0400 RAM_BASE
0x03FF ROM_END
0x0000 ROM_BASE
```

Program outline:

```
BEGIN
  FOR i = length of program DO
    LOAD ROM_BASE + i;
    STORE RAM_BASE + i;
  DONE;

  FOR ever DO
    LOAD DATA_IN;
    MOVE DATA_IN;
    LOAD LUT_TABLE_BASE;
    ADD ;
    LOAD [LUT_TABLE_BASE + DATA_IN];
    STORE DATA_OUT;
  DONE;
END;
```

