Faculty of Engineering, Cairo University

Computer Engineering Department

Computer Architecture Course Project

Orthrus - Phase 1 Report

Ahmad Khaled	Section 1 BN. 03
Maryam Shalaby	Section 2 BN. 21
Zeinab Rabie	Section 1 BN. 22
Omnia Zakaria	Section 1 BN. 23

Contents

1 Introduction **1**

2 Instruction Set Architecture **1**

3 Design **4**

3.1 Overall System Schematic 4

3.2 Memory Unit 4

3.3 Fetch Stage 4

3.4 Decode Stage 5

3.5 Execute Stage 5

3.6 Memory Stage 5

3.7 Write Back Stage 5

3.8 Hazard Detection Unit 5

3.9 Branching 5

1 Introduction

Orthrus is a pipelined static dual-issue microprocessor implementing a RISC ISA similar to the MIPS ISA. In the following sections we outline the instruction format, the general design of the processor, as well as the pipeline stages design.

2 Instruction Set Architecture

The structure for the IR for two-operand instructions is given in Figure 1. The structure for one-operand instructions is given in Figure 2. The structure for branching instructions is given in Figure 3. The structure for the rest of the instructions is given in Figure 4.

Direct	Register	Auto-increment	Auto-decrement	Indexed
Code	000	001	010	011
Indirect	Register	Auto-increment	Auto-decrement	Indexed
Code	100	101	110	111

Table 1: Addressing Mode Codes

	R_0	R_1	R_2	R_3
Code	000	001	010	011
	R_4	R_5	R_6?	R_7
Code	100	101	110	111

Table 2: Register Addressing Codes

Instruction	MOV	ADD	ADC	SUB	SBC	AND	OR	XNOR	CMP
OP Code	1111	1110	1101	1100	1011	1010	1001	1000	0111

Table 3: Two-Operand Instruction Codes

Figure 1: IR Structure For Two-Operand Instructions

15 — 12	11 — 9	8 — 6	5 — 3	2 — 0
Instruction	Source Address	Source Register	Destination Address	Destination Register

Instruction specifies the instruction to execute. Possible codes given in Table 3.

Source Address specifies the addressing mode to choose for the source. Possible codes given in Table 1.

Source Register specifies the source register. Possible codes given in Table 2.

Destination Address specifies the addressing mode to choose for the destination. Possible codes given in Table 1.

Destination Register specifies the destination register. Possible codes given in Table 2.

Instruction	INC	DEC	CLR	INV	LSR	ROR	RRC	ASR	LSL	ROL	RLC
OP Code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010

Table 4: Single Operand Instruction Codes

Figure 2: IR Structure For Single-Operand Instructions

15 — 12	11 — 8	7 — 6	5 — 3	2 — 0
0 1 1 0	Instruction	0 0	Operand Address	Operand Register

Instruction specifies the instruction to execute. Possible codes given in Table 4.

Operand Address specifies the addressing mode to choose for the operand. Possible codes given in Table 1.

Operand Register specifies the operand register. Possible codes given in Table 2.

Instruction	BR	BEQ	BNE	BLO	BLS	BHI	BHS
OP Code	000	001	010	011	100	101	110

Table 5: Branching Instruction Codes

Figure 3: IR Structure For Branching Instructions

15 12	11 9	8 0
0 1 0 1	Instruction	Operand

Instruction specifies the instruction to execute. Possible codes given in Table 5.

Operand specifies the branching offset.

Instruction	JSR	RTS	HITR	IRET	HLT	NOP
OP Code	0100	0011	0011	0010	0001	0000

Table 6: Miscellaneous Instruction Codes

Figure 4: IR Structure For Miscellaneous Instructions

15 12	11	10 0
Instruction	Extra	Operand

Instruction specifies the instruction to execute (except for differentiating RTS and HITR, done using Extra bit). Possible codes given in Table 6.

Extra useful for JSR and HITR only. specifies whether the Operand is given directly or using a register/addressing mode for JSR. Indicates HITR when instruction is 0011 and Extra= 1, indicates RTS if instruction is 0011 and Extra= 0.

Operand useful for JSR only. specifies the memory location to jump to.

3 Design

3.1 Overall System Schematic

Figures 5 and 6 show the overall design of Orthrus, along with the main connections. Note that some of the connections specific to hazards and forwarding are omitted for clarity, and that some of the connections between pipeline register buffers are also omitted for clarity (but if two registers share the same name across two register buffers then they should be connected). Note as well that our design **has no central control unit**. The units in each stage are responsible for generating control signals for themselves given the inputs from the previous stage, from the memory, and from the Hazard/Forwarding unit. In the following subsections, we discuss each unit shown in the diagram in detail. Wemostly supress the details of hazard handling by assuming that all the units mentioned always have a **stall** input and a **flush** input that cause the unit to stall its corresponding stage (by propagating a No-Op to the next instruction) or flush whatever instruction is in it (by propagating a No-Op to the next instruction and having the previous pipeline buffer reset). A more thorough discussion of hazards is given in the Hazard unit subsection.

3.2 Memory Unit

The memory block used is interfaced by two address lines, with the second line given priority whenever a read / write is requested by it. A single word is read or written when an address is on Address2. When an address is on the first line, **two words** are read at a time starting from the word at the address given by Address1. The first line is only for reading from the memory and has no write option. The read operation is instantaneous while the write operation takes 1 clock cycle (the value is written onto the memory on the falling edge of the clock).

3.3 Fetch Stage

In the fetch stage, the Program Counter (PC) register is connected to Address1 of the memory, the data is then read into the fetch and pre-decode unit which has three outputs: the IRs (Instruction Registers) corresponding to the two streams in Orthrus, IR1 and IR2, as well as the increment to add to the PC for fetching the next instruction. The key point is that the pre-decode unit must do not let two memory-accessing instructions simultaneously into the two streams (since this would result in a structural hazard) and hence passes a No-Op into the second stream when this is detected. The pre-decode unit also passes a No-Op into the second stream when an immediate instruction is loaded, since an immediate instruction takes two words. The pre-decode unit must also pass a No-Op when the two instructions try to use the output port. To reduce hardware complexity, a CALL must always be in the first stream, and a No-Op is inserted to fix that if it is not the case. The flow chart for the unit's operation is given by Figure 7. For the flow chart, the instructions that access memory are: PUSH / POP / LDD / STD / CALL / RET / RTI. In addition to the pre-decoding unit mentioned, the fetch stage also includes an incrementor to increment the PC as well as a small unit (shown in Figure 8) to choose whether to get the PC from the incremented PC, normal branches, or from RET / RTI / ITR (which are executed in the last stage by necessity of fetching from memory).

3.4 Decode Stage

3.5 Execute Stage

3.6 Memory Stage

3.7 Write Back Stage

3.8 Hazard Detection Unit

3.9 Branching

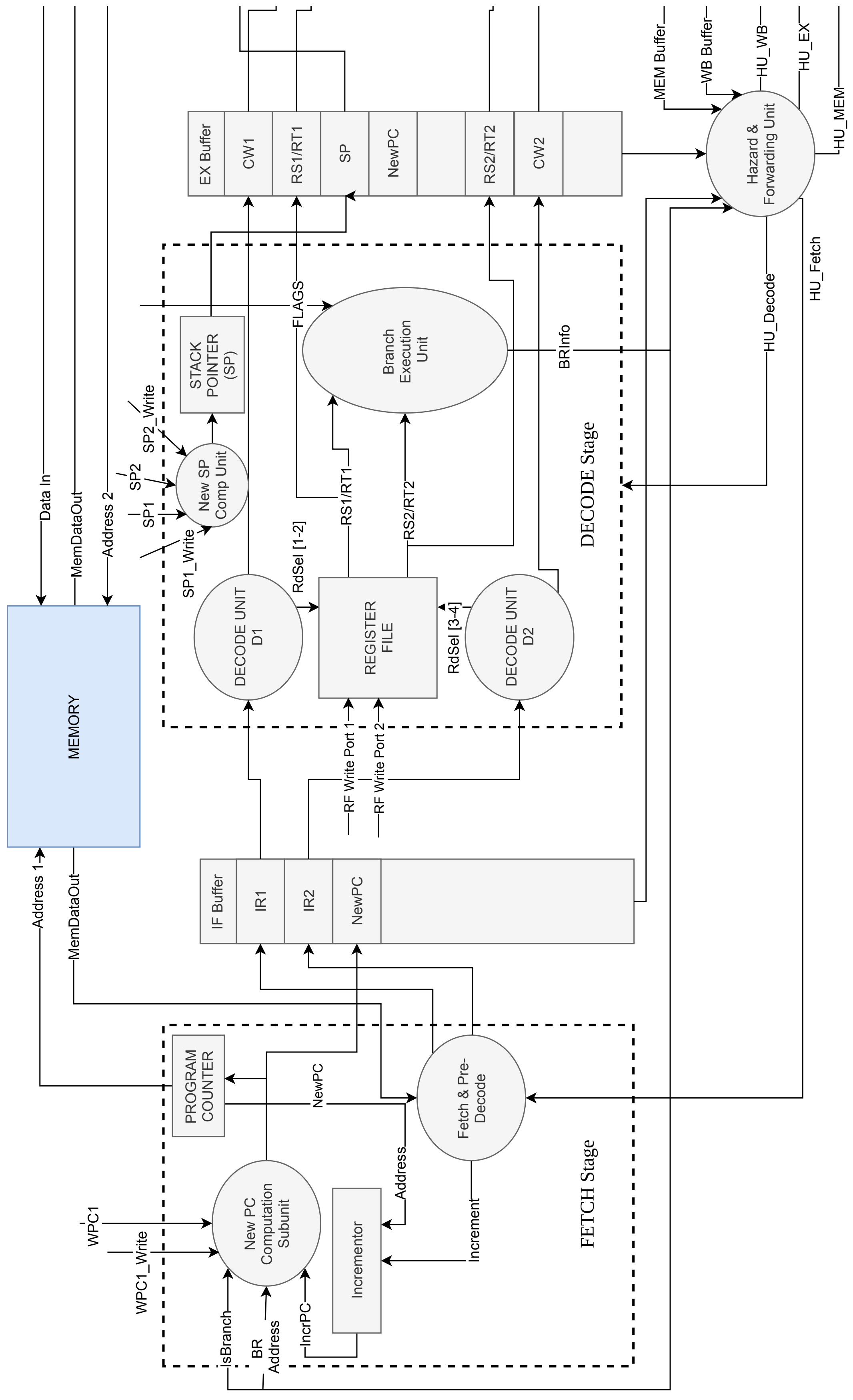

Figure 5: Overall Schematic Part 1

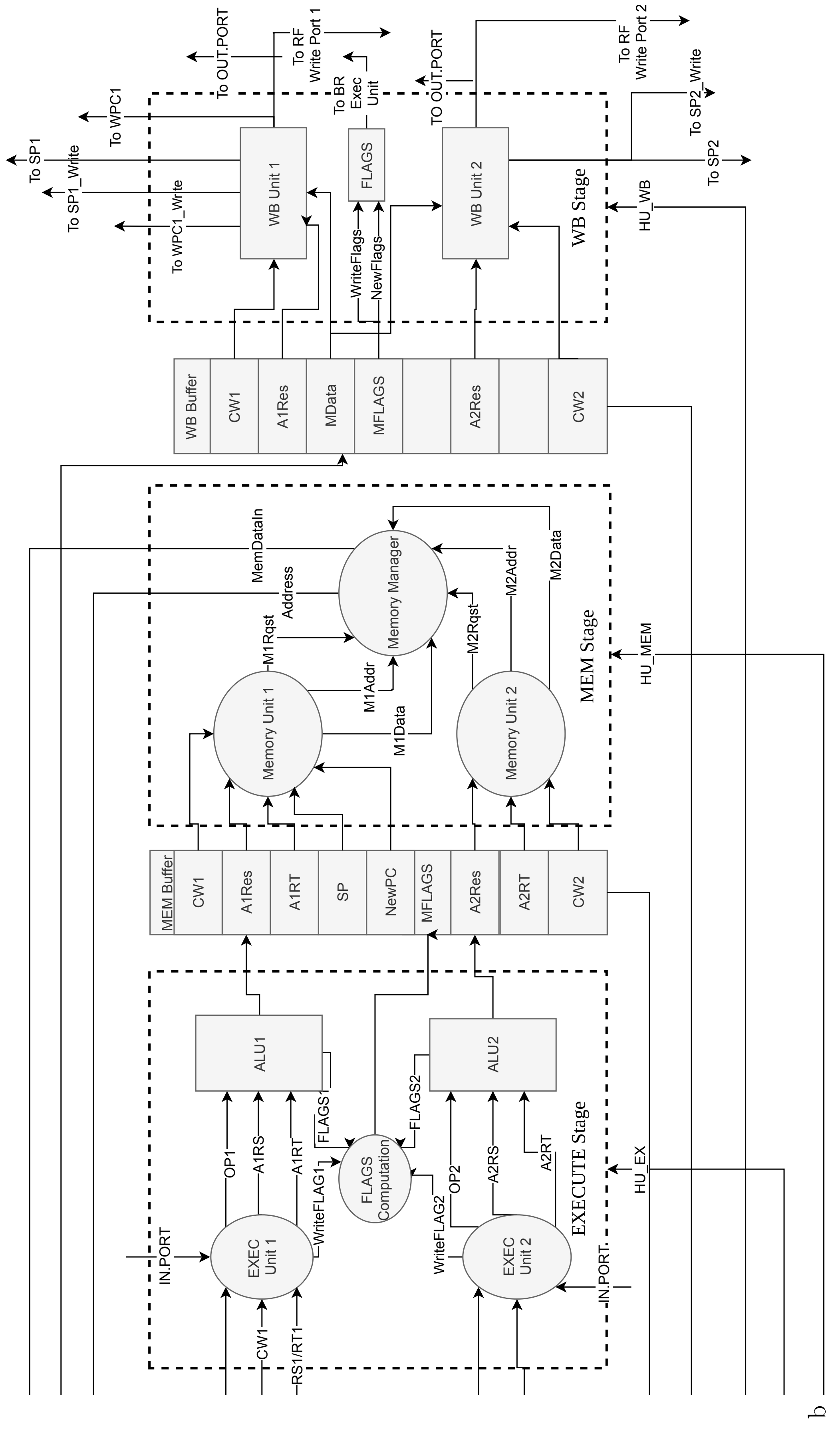

Figure 6: Overall Schematic Part 2

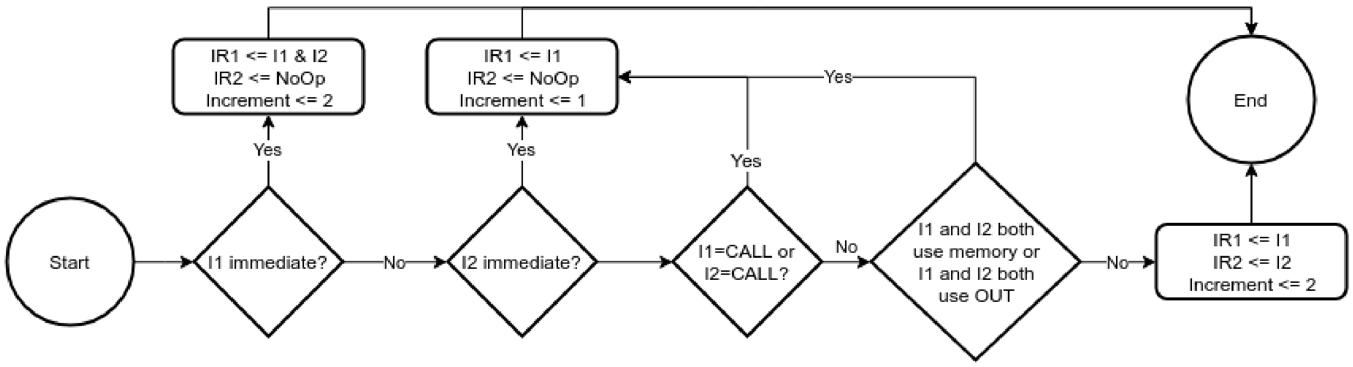

Figure 7: Fetch and Pre-Decode Unit Operation

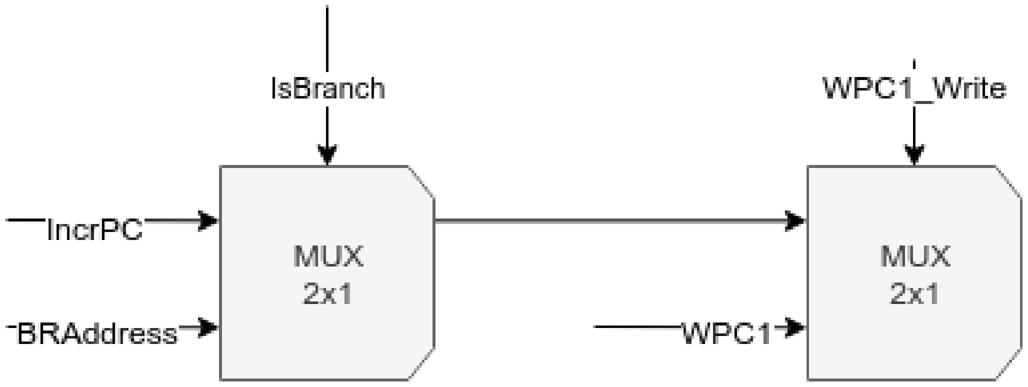

Figure 8: Next PC MUXing Unit