COMP22111: Processor Microarchitecture

The Stump Instruction Set Overview

Stump is a 16-bit RISC, load/store architecture processor developed for teaching purposes by Dr. D. A. Edwards & Dr. A. Bardsley at the University of Manchester. It has eight 16-bit registers plus four separate status flag bits, as illustrated in Figure 1, where the status flags are used to record the status of various arithmetic and logical operations. Register 0 (R0) is not writeable and always reads 0000. Register 7 (R7) acts as the Programme Counter (PC), which is incremented by 1 after each instruction fetch, and is set to 0000 on reset. The memory is 16 bits wide, so 2^{16} = 65536 (FFFF) words may be addressed, as illustrated in Figure 1.

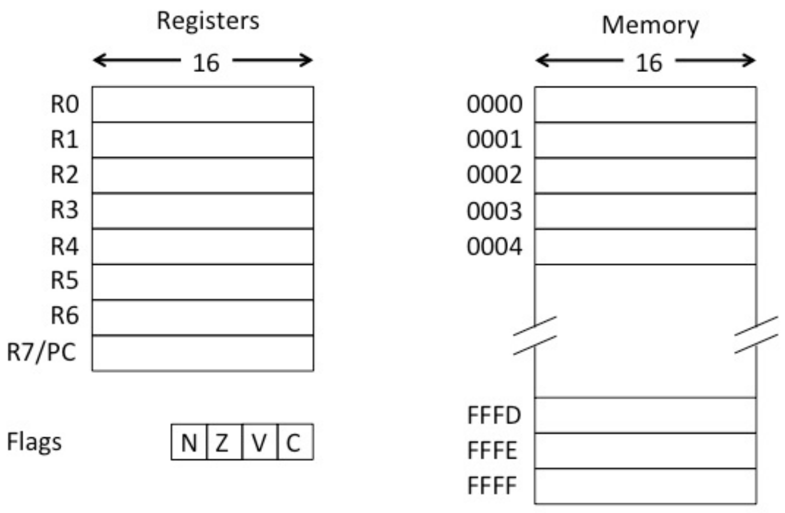

Figure 1: Stump Registers & Memory Map

The instruction set is loosely based on the ARM instruction set, although much simpler. The Stump instructions are 16-bits with three instruction formats, as illustrated in Figure 2.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type 1	instr			0	ST CC	dest			srcA			srcB			shift	
Type 2	instr			1	ST CC	dest			srcA			immediate				
Type 3	1	1	1	1	condition				offset							
reserved	1	1	1	0	-											

Figure 2: Stump Instruction Formats

The first two formats, type 1 and type 2, are very similar, with each having three operands. They are distinguished by a difference in bit 12, which identifies the different addressing modes, i.e. whether the second operand is specified as a register, or as a 5-bit, 2’s complement immediate value (which must be sign-extended to 16-bits inside the processor). In both cases the operation is specified by bits 15:13. The register destination (or, in the case of ST, the data source) is specified by bits 10:8. The first source register is specified by bits 7:5. Sign-extension means a 5-bit immediate value code for (decimal) 2’s complement numbers -16 (10000) to +15 (01111) is extended to 16-bits for the required operation.

Apart from the different addressing modes, another difference between Type 1 and Type 2 instructions is that in the Type 1 instruction a shift operation can be applied to the contents of the register specified by *srcA*, which will be described later.

The ‘*instr*’ code specifies one of six data operations or a memory transfer or a conditional branch. The available instructions are illustrated in Figure 3.

15	**14**	**13**		
0	0	0	ADD	Two’s complement add
0	0	1	ADC	Two’s complement add with carry in
0	1	0	SUB	Two’s complement subtract
0	1	1	SBC	Two’s complement subtract with borrow
1	0	0	AND	Bitwise AND
1	0	1	OR	Bitwise OR
1	1	0	LD/ST	Memory transfer
1	1	1	Bcc	Conditional branch

Figure 3: Stump Instructions

All data operations are similar: a register (*srcA*) and either another register (*srcB*) or an immediate value are fed through an ALU with the result written back to another, independently specified register (*dest*). In the first case the remaining two bits of the instruction can specify a shift operation on *srcA*. Bit 11 is used to specify if the status flags are to be affected (1) or not (0). In the case of a LD or ST instruction an address is calculated in the execute phase, which is written to a separate address register, and not back to the register bank. In this case the memory access is performed in a separate memory phase.

Stump has a condition code register that contains four flags:

N - the **N**egative flag (bit 3) is set (to ‘1’) if the ALU result is negative (when interpreted as a two's complement number) and clear (‘0’) otherwise,
Z - the **Z**ero flag (bit 2) is set if the result is zero, otherwise clear,
V - the o**V**erflow flag (bit 1) is set if the result from an addition/subtraction interpreted as a two's complement number is ‘wrong’, otherwise clear,

C - the **C**arry flag (bit 0) is set if there is a carry out from the most significant bit of the result (bit 15), otherwise clear.

The status flags represent the status of an ALU calculation. In Stump the ALU always generates a set of flags, however, the condition code register is only updated when the instruction mnemonic has an 'S' appended to it, such as ANDS, SUBS, ANDS etc. This is indicated by the status of bit 11, which is 1 when the condition code register needs updating.

The N and Z flags are easy to derive and apply to both arithmetic and logical functions. The V and C flags are only relevant for arithmetic operations. Some more detailed examples of flag derivation are given in the description for exercise 2.

Memory transfers (*instr* = 110) never affect the flags. In this case, bit 11 is used to determine the direction of a transfer, so for a LD instruction, bit 11 = 0, whereas for a ST instruction bit 11 = 1. In the case of a LD instruction, *dest* is the destination for the data loaded from the memory address. In the case of a ST instruction, *dest* is the location of the data to be stored to the memory address. In both cases an address is calculated during the execute phase using an ADD instruction – the addressing mode used depends on whether the instruction is Type 1 or Type 2.

Type 3 (or conditional branch) instructions are specified differently. Here, bit 12 is always '1' and bits 11:8 specify the branch condition. The remaining 8 bits form an 8-bit signed offset from the PC (R7). As the offset is sign-extended to 16-bits before being added to PC branches can be forward or backward by a number of words. A list of the conditional branch instructions used in Stump is given in Figure 4.

Mnemonic	Bits 11:8	Branch Condition	Notes
BAL	0000	Always	
BNV	0001	Never	
BHI	0010	$C + Z = 0$	comparison: unsigned arithmetic
BLS	0011	$C + Z = 1$	
BCC	0100	$C = 0$	overflow test: unsigned arithmetic
BCS	0101	$C = 1$	
BNE	0110	$Z = 0$	zero test
BEQ	0111	$Z = 1$	
BVC	1000	$V = 0$	overflow test: signed arithmetic
BVS	1001	$V = 1$	
BPL	1010	$N = 0$	comparison: signed arithmetic
BMI	1011	$N = 1$	
BGE	1100	$\bar{N}.V+N.\bar{V}=0$	
BLT	1101	$\bar{N}.V+N.\bar{V}=1$	
BGT	1110	$(\bar{N}.V+N.\bar{V})+Z=0$	
BLE	1111	$(\bar{N}.V+N.\bar{V})+Z=1$	

Figure 4: Stump Conditional Branch Instructions

So, in the case of a Type 3 instruction a new address is calculated by adding the offset to the PC, which is only written back to the PC if the branch condition is satisfied, i.e. the branch should be taken. This implies that the PC will have already been incremented and is already **pointing to the next instruction**. Thus, a branch with offset 00 will act as a no-op. To branch to the branch instruction itself requires an offset of -1 (coded as FF) - and so on. 'Correcting' for this would cost extra hardware so it is typical to leave such 'features' when specifying the instruction coding, and the required offset is calculated automatically by the an assembler.

A Verilog 'task' is provided (see 'Testbranch' in Stump_control.v) to provide a go/no-go comparison from the flags and condition code so you don't have to work out whether a branch should be taken or not ('Testbranch' returns true '1', or false '0' depending on whether the branch should be taken, or not). However, a glance at the simpler ones may be beneficial.

Shift operations on Stump are very limited and somewhat of an afterthought. The 2-bit code to specify the shift operation on Type 1 instructions allow 4 operations, these are listed in Figure 5. The 'no shift' case is the default unless a shift is specified explicitly.

Code	Shift	Action
00	-	No shift applied
01	ASR	Arithmetic shift right
10	ROR	Rotate right
11	RRC	Rotate right through carry

Figure 5: Stump Shift Operations

What is the carry out from the shifter?

The carry out from the shifter, denoted csh, is only used for logical operations when the status register update bit is set, i.e. instructions ANDS and ORS. In the case of a logical instruction with a shift (Type 1) then the carry flag in the status register should be set to the value of the carry out from the shifter. In the case of a Type 1 logical instruction with no shift, or a Type 2 logical instruction, then the carry flag must be set to zero, which can be used as a means to clear the carry flag. (This is reminiscent of the behaviour of ARM processors.)

A possible Stump microarchitecture

Figure 6 illustrates a suggested implementation of the Stump that is fairly straightforward, although not the fastest in terms of performance. This is the same RTL view of the Stump processor you have been exposed to in the lectures.

- 'Registers' contains the architectural (programmer-visible) registers R0-R7; the other programmer-visible state is in the status flags in the condition code register (CC Reg).

- The Instruction Register (IR) and the address register (Address Reg) are added by the architect to provide temporary state-holding. More complex processors - or more complex implementations of a Stump - may need more 'hidden' registers.
- The other blocks are combinatorial.
- The sign extender will sign extend immediate values for Type 2 and Type 3 instructions.
- Control signals are omitted from this diagram, as is the control block that contains the Stump FSM and the combinatorial logic for generating the system control signals.

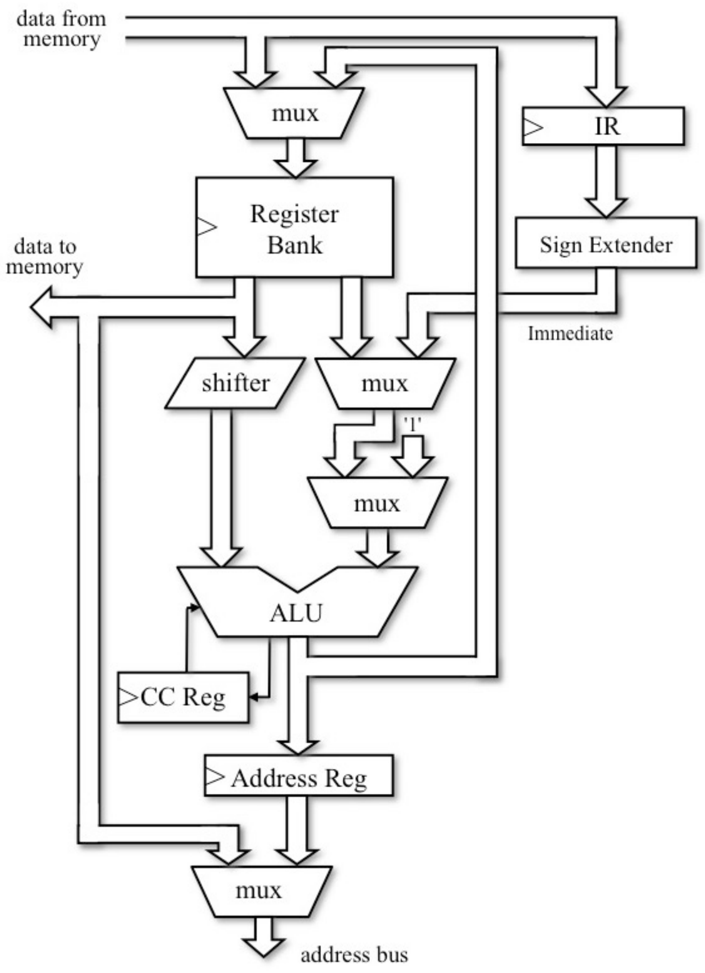

Figure 6: Architectural view of the Stump processor datapath

By the end of this laboratory you will have implemented a Stump processor and demonstrated it working on the experimental boards in the laboratory. The complete processor will be implemented in Verilog, with the architecture shown in Figure 6 being implemented as structural Verilog. Some elements of the datapath, such as the register bank, multiplexers, and registers have been provided for you as Verilog modules. Other elements of the datapath, such as the ALU and control block (not shown) you will have to create from the template modules provided. In addition, as when designing any hardware, you will be expected to simulate your designs in order to confirm that they work to the required specification. Test! Test! Test!