# COMP22111: Processor Microarchitecture

## The Stump Processor ISA 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 – an instruction can write to R0 but it will have ne effect.

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.

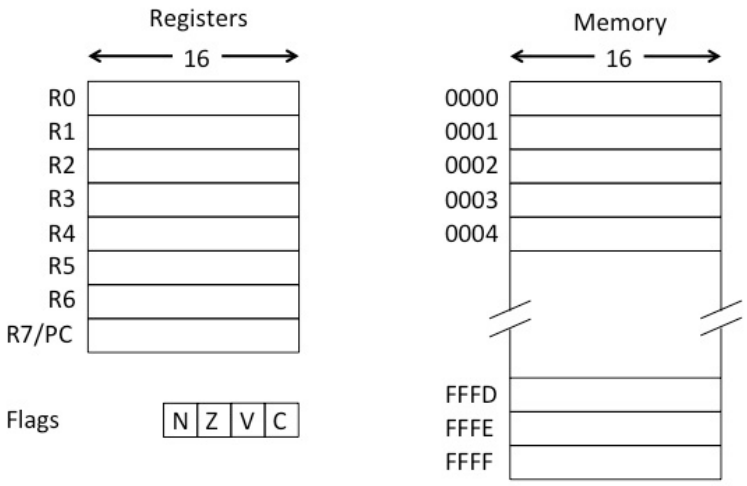


Figure 1: Stump Registers & Memory Map

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 instruction formats, type 1 and type 2, are very similar, with each having three operands. They are distinguished by bit 12, which identifies the different addressing modes, i.e. whether the second operand is specified as a register for Type 1 instructions, or has a 5-bit, 2's complement immediate value for Type 2 instructions. 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 and in the case of Type 1 instructions the second source register is specified by bits 4:2. In the case of Type 2 instructions, the 5-bit immediate value, specified by bits 4:0, must be sign extended to 16-bits before being used in an operation, this is achieved by simply copying the sign bit from bit 4 in to bits 5:15.

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 (Type 1 and 2 instructions) or a conditional branch (Type 3 instructions). The available instructions are illustrated in Figure 3.

| **15** | **14** | **13** | | | Type |
|---|---|---|---|---|---|
| 0 | 0 | 0 | ADD | Two's complement add | 1 or 2 |
| 0 | 0 | 1 | ADC | Two's complement add with carry in | 1 or 2 |
| 0 | 1 | 0 | SUB | Two's complement subtract | 1 or 2 |
| 0 | 1 | 1 | SBC | Two's complement subtract with borrow | 1 or 2 |
| 1 | 0 | 0 | AND | Bitwise AND | 1 or 2 |
| 1 | 0 | 1 | OR | Bitwise OR | 1 or 2 |
| 1 | 1 | 0 | LD/ST | Memory transfer | 1 or 2 |
| 1 | 1 | 1 | Bcc | Conditional branch | 3 |

Figure 3: Stump Instructions

For all Type 1 and Type 2 instructions (so all apart from conditional branch instructions which are Type 3) all data operations are similar: a register (***srcA***) and either another register (***srcB***) or an immediate value (from the instruction and sign extended) are fed through to the ALU with the result written back to another, independently specified register (***dest***). In the case of Type 1 instructions 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) for all Type 1 and Type 2 instructions apart from LD/ST instructions. In the case of a LD/ST instructions bit 11 is used to specify if the instruction is LD (bit 11 = 0) or ST (bit 11 = 1) as both instructions have the same bit code. In LD/ST instructions 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, a separate 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.

Memory transfers, i.e. LD/ST, *instr* = 110, never update the flags, as, 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.

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 it to the assembler to determine the required offset. In the code, labels should be used to specify the location of the instruction to jump to.

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, based on the status of the flags in the condition code register).

Shift operations on Stump are very limited and somewhat of an afterthought. The 2-bit code to specify the shift operation in 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 in the assembly code.

| 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

| 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

## 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.

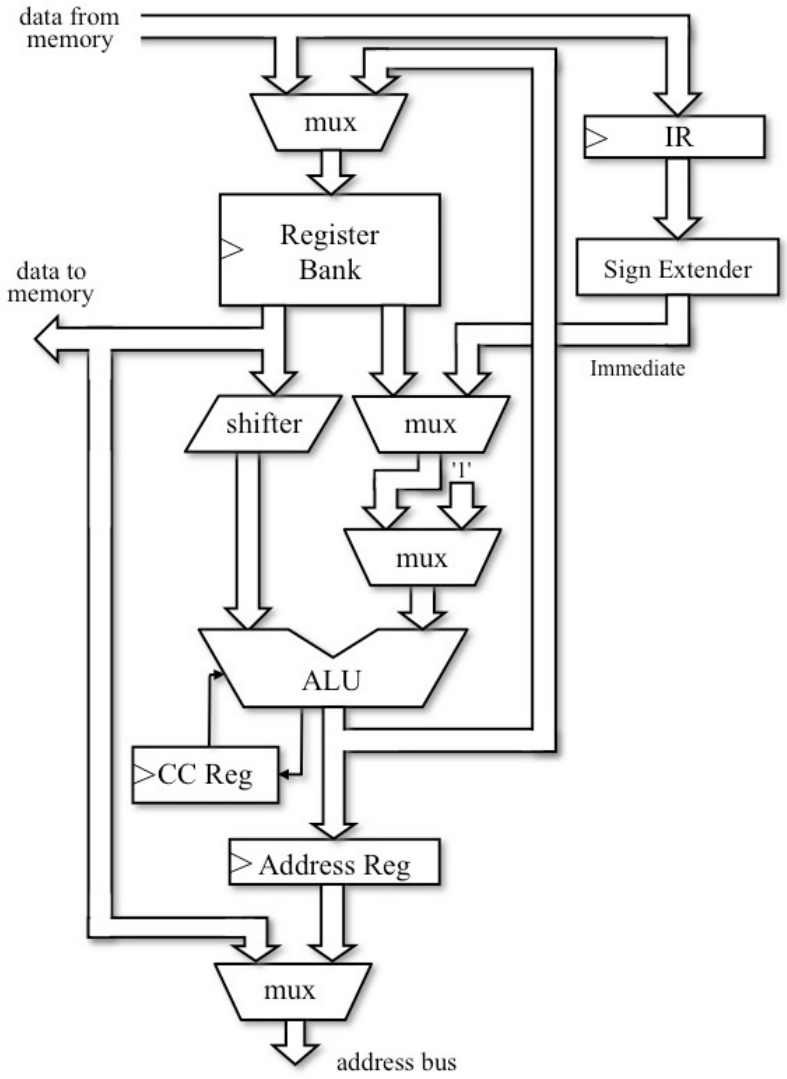


Figure 6: Architectural view of the Stump processor datapath