1 Introduction

This report details the research done for the ‘Test and Branch’ part of team R4 for the VLSI design module (ELEC6027). It looks into 2 aspects of processor design - flags and conditional branches. In this report, three case studies are documented, explaining how they implement conditional branches, and which flags they use, if any. Other architectures of note are also discussed. The report then finishes with the conclusions that can be drawn from this research. The authors’ recommendation for the implementation is also given in the final section.

2 Case studies

Three case studies are discussed here. For each architecture studied, there is a description of the flags implemented (if any) and instructions used for conditional branches. Also, two C code snippets, seen in listings 1 and 2, are converted to assembler for each architecture to see how they compare.

Listing 1: C Code

```c
uint16_t a = 0;
for(uint16_t i = 0; i < 10; i++)
{
        a = a + i;
}
```

Listing 2: C Code

```c
if ( a > 0 )
        b = 1;
else
        b = 0;
```

2.1 ARM Cortex M0

The ARM Cortex M0 is a 32-bit RISC architecture that implements the Thumb/Thumb2 instruction set [1]. It implements 19 formats of instructions in total, one of which is the ‘conditional branch’ instruction. The conditional branch instruction depends on the value in the status register from a previous operation.

Table 1: Explanation of the flags in the ARM Cortex M0

Flag	Shorthand	Explanation
Carry	C	Set on arithmetic carry or borrow
Overflow	V	Indicates two's complement overflow
Zero	Z	Set if the ALU result is zero
Negative	N	Set to the most significant bit of the result (sign).

2.1.1 Flags

The Cortex M0 uses 4 flags; Carry, oVerflow, Zero and Negative [2]. A brief description of them can be seen in table 1. The flags are stored in a specific register, the Application Program Status Register (APSR). The APSR is updated by a number of instructions. All arithmetic instructions have the ability to update the register depending on the result of the arithmetic operation. This can be seen in the assembler by an 'S' suffix to the instruction. This optional update is only in the Thumb2 instruction set, not the Thumb. In the Thumb instruction set, all ALU operations update the status register.

There are four other instructions which update the status register - Compare (`CMP`), Compare Negative (`CMN`), Test Bits (`TST`) and Test Equivalence (`TEQ`). These can compare two values from registers, or a register and an immediate [3]. Compare subtracts the register value from the other operand, and `CMN` is the same, but negates the second operand. These two instructions update all the flags, but discard the result of the operation. `TST` performs an AND function between the two operands. This is the same as an `AND` instruction, but only affects the `Z` and `N` flags. Similarly, there exists a `TEQ` instruction which conducts an XOR between the two operands and only updates the `Z` and `N` flags [4]. The `TST` and `TEQ` instructions do not store the result of the operation.

2.1.2 Conditional Branch

The Thumb instruction set has a conditional branch instruction. This is capable of incrementing the program counter by an immediate value, depending on the flags in the status register. Fourteen different conditions are supported, and the full list can be seen in [5]. They include individual flag tests, as well as both signed and unsigned greater/less than tests. The value of the flags is from the last instruction that set the flags.

The Thumb2 instruction set also allows for conditional execution. However, this will not be discussed within the scope of this report.

2.1.3 Assembler Examples

Below are snippets of assembly language for the Cortex M0 to implement the C code seen in listings 1 and 2.

Listing 3: Thumb/Thumb2 Assembler for code in listing 1

```
.def a=$r1
.def i=$r2
MOV a #0 ;load a with 0
MOV i #0 ;load i with 0
loop:
CMP i #10 ;compare i to 10.
BEQ exit ;branch if i == 10
ADD a a i ; a = a + i
ADD i i #1 ; i++
B loop ; jump to loop
exit:
...
```

Listing 4: Thumb/Thumb2 Assembler for code in listing 2

```
.def a $r1
.def b $r3
CMP a #0 ; compare a to 0
BNE else ; if a != 0 go to else
MOV b #1 ; if a == 0, load b with 1
B exit
MOV b #0 ; if a != 0, load b with 0
exit:
```

2.2 Intel 8086

The 8086 is a 16 bit microprocessor released in 1978. This was the original processor designed by Intel. It is a CISC based architecture and implements the original x86 instruction set. Since then, the x86 instruction set has been expanded and the original instructions are still implemented in the lasted Intel CPUs.

2.2.1 Flags

The Intel 8086 has 9 flags, detailed in table 2. The status register is updated when a compare (`CMP`, `CMPSB`, `CMPSW`) instruction is executed, or after an

Table 2: Intel 8086 Flags from [6]

Flag	Shorthand	Explanation
Carry	C	Carry from the 8 bit arithmetic
Zero	Z	Set if the result of the ALU is zero
Sign	S	Set to the most significant bit of the ALU result
Overflow	O	Set on two's complement overflow
Parity	P	1 if the ALU result is even parity, 0 if odd [7]
Auxiliary (or Adjust)	A	The carry generated from the low 4 bits of the ALU operation [8]
Trap	T	Used for single step debugging [9]
Direction	D	Sets the source index to increment / decrement (0/1) [10]
Interrupt	I	Enables / disables interrupts [11]

arithmetic operation. There are flags which are not used for conditional branches, but related to other workings of the processor, and hence are only mentioned in passing in this report.

2.2.2 Conditional Branch

The 8086 has a total of 31 conditional branch instructions, and one jump instruction [12]. The conditional branches cover many different logic combinations between 5 flags - `C`, `Z`, `O`, `S` and `P`. This enables the 8086 to make signed and unsigned decisions. The full set of branch instructions can be seen in [13]. There is also a branch instruction that checks that the value of register `CX` is zero, rather than the flags.

In the assembly language, there are multiple instructions that map to the same opcode. For example, the `JE` instruction checks that the `Z` flag is set, but the `JZ` instruction also does this. This is done to make the code more readable whilst keeping a small opcode size [14]. In total, therefore, there are 17 actual instructions.

The 8086 also implements a `LOOP` instruction. This is linked to a specific register (the `CX` register) and will branch if the value of `CX` does not become 0 after being decremented. There is another instruction, `LOOPE / LOOPZ`, that jumps if the counter is zero, and the zero flag is set. This has a complementary instruction, `LOOPNE / LOOPNZ`, which jumps if the counter is nonzero and the zero flag is not set. These instructions are useful, as it combines the decrement of a counter with a branch, resulting in a quicker operation for loops.

2.2.3 Assembler Example

Listing 5: Intel 8086 assembler for listing 1

```
.def a ax
.def i cx
mov a,0 ; initialise a
mov i,10 ; initialise i
looplabel:
add a,a,i ; do the sum
loop looplabel ; cx--, if cx != 0, jump to looplabel
```

Listing 6: Intel 8086 assembler for listing 2

```
.def a ax
.def b bx
cmp a,0 ; compare a to zero
je else: ; jump if zero flag is set (0 == a)
mov b,0 ; store 0 to b
jmp exit ; have to jump to exit
else:
 move b,1 ; store 1 to b
exit:
```

2.3 MIPS

MIPS is a 32-bit RISC architecture. It is one of the early RISC machines, now owned by Imagination Technologies and is still used today [15]. The simple design of the MIPS architecture lends itself as a learning tool for computer architecture.

2.3.1 Flags

MIPS does not have a status register, and only one flag. The flag used is a zero flag. This means that the conditional branches can only rely on whether a register value is zero or not.

2.3.2 Conditional Branch

MIPS implements only two branch instructions; branch if equal (`BEQ`) and branch if not equal (`BNE`). With only these two instructions, it is difficult to see how this can be used to fully control the program flow. There exists an instruction which helps with control flow, but is not a conditional jump : the Set Less Than (`SLT`) instruction. `SLT` compares two register values. A one is stored in the destination register if operand two is less than operand three, else zero is stored [16].

This instruction fulfils the lack of branch instructions. MIPS also has a register which always reads zero, which can then be compared to the value returned by the `SLT` instruction. By not having flags, more instructions need to be executed to conduct a branch. However, the instruction set on a whole will be smaller.

2.3.3 Assembler Example

Listing 7: MIPS assembler for listing 1

```
.def a $r1
.def i $r2
add a $r0 $r0 ; load a with 0
add i $r0 0 ; load i with 0
loop:
slti $r3 i 10 ; slt with immediate - i < 10
beq $r0 $r3 exit ; if r3 == 0, jump to exit
add a a i ; do the add
jmp loop ; jump back to the beginning of the loop
exit:
```

Listing 8: MIPS assembler for listing 2

```
.def a $r1
.def b $r2
slt $r3 $r0 a ; 0 < a
bne $r0 $r3 else ; jump if the result isn't zero
addi b $r0 1 ; load b with 1
jmp exit
else:
addi b $r0 0 ; load b with 0
exit:
```

3 Other Architectures

3.1 DEC Alpha

Architectures that don't use flags are not common. The DEC Alpha is another example of an architecture that does not use flags. The conditional branch instruction can conduct six comparisons to zero; equal, not equal, greater than (or equal) and less than (or equal). The arithmetic for the operation must be done beforehand, as in the MIPS. The conditional branch then can execute by comparing a precomputed value [17].

3.2 AVR

Even though the AVR is a microcontroller core, it still posed interesting reading. The AVR uses a status register and has 8 flags in total [18]. A number of these are similar to ARM and Intel (`C`, `Z`, `N`, `V`, `S`, `H`, `I`), but the AVR implements a `'T'` flag. The `T` flag in the AVR is not the same as in the 8086, but is used as a more user defined bit in the status register. The instruction set allows branching if this flag is set or not, and can be set to a bit value in a byte. [19] has some examples of how this `T` flag can be used.

4 Conclusion

There is a compromise that must be made with the test and branch aspect of processor design. The use of flags is advantageous as more complex branches can be conducted depending on the result of the ALU operation. Also, the branch only takes one instruction to complete the comparison, and one to do the jump, totalling two in the worst case. However, this then requires more instructions, potentially increasing the size of the opcode field in instruction set. This also requires a more complex design in the ALU to calculate the flags, as well as a more complex control unit to process the data.

Without flags, the compiler / programmer must conduct operations to reduce the branch problem to an equality, or a less-than. Although the example in listings 7 and 8 are simple, this is not always the case. A greater/less than or equal comparison will require addition to an operand, before conduction the `SLT` instruction. This can lengthen the execution of branches, which are used a lot in programs. The advantage, however, is that the hardware implementation is much simpler, and the control is smaller. A compare to zero is needed, as well as an extra ALU operation (`SLT`) which transfers the borrow signal to the LSB of the result. Implementing a comparison instruction, which updates the flags but does not store the result would also be advantageous.

Amdahl's Law says that to increase the performance, the common case must be made fast [20]. Branches are reasonably common to occur in a program, as the principle of a processor is to be able to dynamically make decisions. Performance is a key aspect to processor design and by executing fewer instructions to do the same task will give an increase in performance. The recommendation, therefore, is that flags should be used at the expense of more hardware, instructions and complexity of the control. However, the increased performance and flexibility of the processor should be advantageous.

Word Count: 1569