

CSCE 313

Embedded Systems Programming

2005/2/15

Lectures 16 & 17

M68000: Instruction Codes and Timing

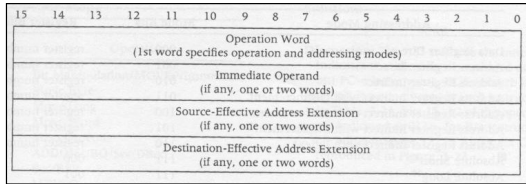
Figures: MacKenzie © 1995 Prentice Hall Publishers, Inc.

Lectures 16 & 17 - Outline

- Objectives.
 - ✓ Understanding these additional elements of the 68000 architecture:
 - ✦ *Instruction formats* – going from the textual instruction specification to the actual hexadecimal or binary sequence of bits read by the processor (hand assembly) and reading machine codes and determining the instruction (disassembly). Appendices B & C.
 - ✦ *Instruction timing* – knowing how many clock cycles a given instruction will take to execute, with its selected addressing modes for its operands. This includes the number of cycles to fetch it from memory, decode it, fetch the operands from memory, decode them, and actually execute what is dictated in the instruction. Appendix D.
- Importance.
 - ✓ Debugging programs
 - ✦ Often, you may be required to read the actual contents of memory, or watch the binary bits scroll across the screen of a logic analyzer. You need to have an understanding of the instruction format and bit encoding, to be able to recognize the instructions.
 - ✓ Understanding the behavior of the processor
 - ✦ Often, you can determine a lot about the internal workings of the CPU by looking at how the instruction set is designed. Some processors have well-design, “clean” instruction sets that are intuitive; others have instructions that have different ways to interpret the bits, depending on the type of instruction. Usually, instruction set designers and processor architects will attempt to save “resources” (cycles, register bits, etc.) by “overloading” the meaning of bits, making it important for the programmer to understand the usage to know the essence of how the processor will behave given a specific instruction bit set.
 - ✓ Programming the processor most efficiently
 - ✦ Often, there are many ways to write a “correct” program, but the issue in embedded systems is often how “tight” you can make the code; it should be correct *and* consume the least amount of resources as possible (memory space, register bits, clock cycles, battery power, etc.).

M68K– General Layout of an Instruction

Source: Figure 3-19. MacKenzie © 1995 Prentice Hall Publishers



Bits 15 through 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/ScC/DBcc
0110	Bcc/BSR
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

Source: Table 3-13. MacKenzie © 1995 Prentice Hall Publishers



• Memory Organization

- ✓ A program "segment" has a sequence of instructions located sequentially in memory.
- ✓ The PC Register points to the first word of the instruction to be read from memory to be decoded and executed.
- ✓ A given instruction can be from 1 to 5 16-bit words in length, depending on the number of operands, source & destination of these operands, and how the effective address is specified for the operands.

• Instruction Layout

- ✓ A given instruction is organized as consecutive 16-bit words in memory (in ascending order, starting with the Operation Word).
- ✓ The figure 3-19 in MacKenzie (at left) illustrates the different components of a possible instruction; **note** that it doesn't really represent any specific instruction in memory. So, a given instruction could include any of the 3 fields after the Opcode word.
- ✓ For example, a **MOVE.W #FF44, D0** would have the following: Opcode Word + Immediate operand. So, it would be a two-word instruction.

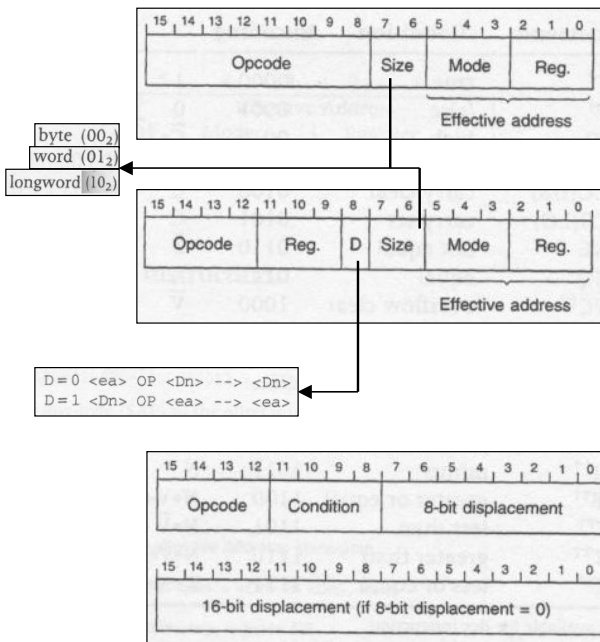
• Instruction Category

- ✓ We use the most significant 4-bits (15 down to 12) to determine the basic "category" of the instruction.
- ✓ From this, we can break apart the remainder of the instruction to determine its meaning.

© 2002 Dr. James P. Davis Page 3

M68K– Instruction Categories & Formats

Source: MacKenzie, Fig 3-20 © 1995 Prentice Hall Publishers



• Single Operand Instructions

- ✓ Examples: CLR.L D0; TST.B D1...etc.
- ✓ Information in the general format: (1) **Opcode** (8-bits), (2) **Size** (Byte, Word Longword, in 2-bits, but value '11' is reserved), (3) **Operand Addressing Mode** (3-bits, encoding 8 possible modes, except when it is '111', where it uses the Register Mode bits to encode an additional 5 mode choices), and (4) **Register Bits** (3-bits, to encode 1 of the 8 address, A0-A7, or 8 data, D0-D7, registers, except when Mode = '111', when the bits in this field are used to encode additional Effective Address or Mode information).

• Double Operand Instructions

- ✓ Examples: MOVE.W \$9000, D0; ADD.L D0, D1...etc.
- ✓ Most 2-operand instructions require one operand to be in a register. The "D" bit (bit 8) indicates whether the register is the source or the destination operand in the instruction. Note: Both operands could be stored in registers, but the D bit is used for most general case.

• Conditional Branch Instructions

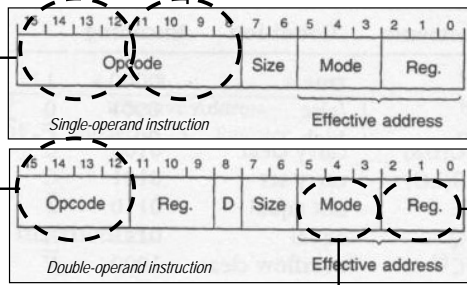
- ✓ Three instruction types: Branch on condition (Bcc), Set by Condition Code (ScC), and Decrement and Branch on condition (DBcc).
- ✓ Examples: BLT \$8040, ST \$9004, DBLE \$8020 (**Note**: most of these would follow one of the Compare CMP instructions, or some other instruction, e.g. SUB, that sets the bits in the CCR).
- ✓ The 8-bit displacement allows "short" branches to be made, if the effective address can be represented in 8-bits; otherwise, this set of 8 bits is zeroed, and the full 16-bit displacement is stored in the second instruction word.



© 2002 Dr. James P. Davis Page 4

M68K Instructions – Format Decoding-1

Source: MacKenzie © 1995 Prentice Hall Publishers



Single-operand instruction format has an 8-bit opcode. What's done with the other 4 bits? It depends on the specific instruction.

For example, NOT (listed as a miscellaneous instruction in Table 3-13), is encoded as '0100' + '0110' in its lower 4 opcode bits.

The Mode and Register fields for the Effective Address (which is either the source or destination operand, depending on bit 'D').

Bits 15 through 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/ScC/DBcc
0110	Bcc/BSR
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

Most-significant 4-bits encodes the instruction category, which is the whole 4-bit opcode for double operand instructions.

Addressing Mode	Mode Bits	Register Bits
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement [†]	101	register number
Address Register Indirect with Index [†]	110	register number
Absolute Short [†]	111	000
Absolute Long ^{††}	111	001
Program Counter with Displacement [†]	111	010
Program Counter with Index [†]	111	011
Immediate or Status Register ^{†††}	111	100

[†]One extension word required
^{††}Two extension words required
^{†††}For Immediate addressing, one or two extension words required depending on the size of the operation
^{*}One extension word required; see Table C-3 in Appendix C for the encoding



M68K Instructions – Format Decoding-2

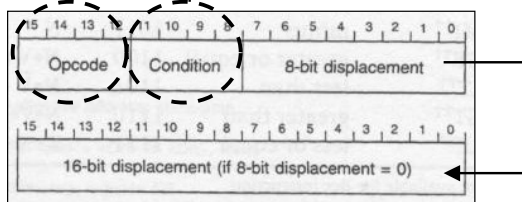
Figures: MacKenzie © 1995 Prentice Hall Publishers

Most-significant 4-bits encodes the instruction category, which is the whole 4-bit opcode for Conditional Branch instructions.

Bits 15 through 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/ScC/DBcc
0110	Bcc/BSR
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

Next 4-bits (11 down to 8) encodes the Condition category, which is indicated in Table 3-12 as the Encoding of the bits. The instruction mnemonic is added to B<>, S<> or DB<> to form the particular instruction which, when assembled, is encoded as shown below, having the indicated meaning. For example, B+NE for BNE for "branch if not equal to zero".

You'll need to memorize these mnemonics, so you'll recognize these instructions when you see them, and know their meaning.



The Displacement is used to calculate either the "jump" address or the location that gets cleared or set (with S<> instruction). If the Displacement field is #0, then CPU will fetch the next word and use it to calculate a 16-bit offset address.

Mnemonic	Condition	Encoding	Test
T [†]	true	0000	1
F [†]	false	0001	0
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LS	low or same	0011	$C + Z$
CC(HS)	carry clear	0100	\overline{C}
CS(LO)	carry set	0101	C
NE	not equal	0110	\overline{Z}
EQ	equal	0111	Z
VC ^{††}	overflow clear	1000	\overline{V}
VS ^{††}	overflow set	1001	V
PL ^{††}	plus	1010	\overline{N}
MI ^{††}	minus	1011	N
GE ^{††}	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT ^{††}	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT ^{††}	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE ^{††}	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

[†] Not available for Bcc instruction
^{††} Twos complement arithmetic, signed numbers



M68K Instruction Decoding – Example 3-1

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-1. Convert the following 68000 assembly-language instruction to machine language:

ADD.W #512,D2

Solution:

D47C (1st instruction word)
0200 (2nd instruction word)

Discussion: The first step is to look up the opcode bits 15–12 in Table 3-13. For ADD instructions, bits 15–12 are 1101₂. The second step is to look up the full opcode in Appendix C, pages 366 to 377. These pages tabulate all 68000 instructions, sorted numerically by the binary code. On page 376, the ADD instruction is found with the encoding shown in Figure 3-21. ■

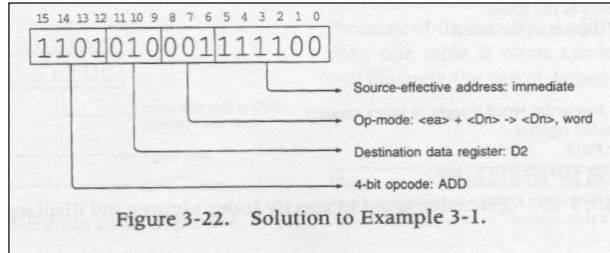


Figure 3-22. Solution to Example 3-1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Data Register				Op-mode		Effective Address					
										Mode		Register			

Op-mode field:

Byte	Word	Long	Operation
000	001	010	<ea> + <Dn> -> <Dn>
100	101	110	<Dn> + <ea> -> <ea>

Figure 3-21. Encoding of ADD instruction.



© 2002 Dr. James P. Davis Page 7

M68K Instruction Decoding – Example 3-2

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-2. Convert the following 68000 assembly-language instruction to machine language. Assume the instruction is in memory location \$2050.

BGE.S \$2000

Solution:

6CAE

Discussion: The “.S” suffix to the mnemonic indicates that the offset is encoded as a short (8-bit) PC-relative value. BGE (branch if greater or equal) is a conditional branch instruction (Bcc). In Table 3-13, bits 15–12 of the operation word are indicated as 0110₂ for Bcc instructions. The encoding on page 373 in Appendix C is reproduced in Figure 3-23. ■

- We'll work this in class:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	Condition [†]				8-Bit Displacement							

[†] See Table 3-12 or Table A-2

Figure 3-23. Encoding of Bcc instruction.



© 2002 Dr. James P. Davis Page 8

M68K Instruction Decoding – Example 3-3

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-3. Convert the following 68000 assembly-language instruction to machine language:

```
MOVE.L 0(A3,D4),D7
```

Solution:

```
2E33 (1st instruction word)
4000 (2nd instruction word)
```

Discussion: From Table 3-13, bits 15–12 of the instruction word are 0010₂. In Appendix C, the complete encoding of the instruction is given on page 368. This is repeated in Figure 3-24 for convenience. ■

- We'll work this in class:

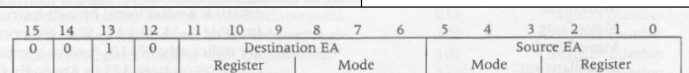
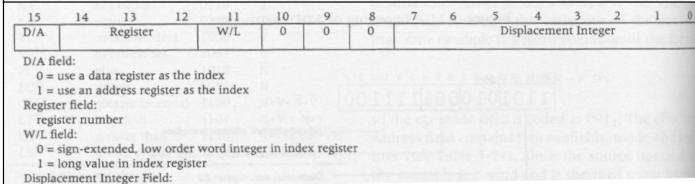


Figure 3-24. Encoding of MOVE.L instruction.



M68K Instruction Encoding – Example 3-5

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-5. Convert the 68000 machine-language instruction 91D4₁₆ to assembly language.

Solution:

```
SUBA.L (A4),A0
```

Discussion: Bits 15–12, 1001₂, identify the category of instruction. In Table 3-13 this is shown as SUB/SUBX. From page 374 in Appendix C the encoding of these instructions is given. A process of elimination is used to choose among the possibilities and determine the size of the operand(s) and the source and destination addressing modes. For example, SUBX has bit 4 = 0 (see page 374). Since bit 4 = 1 in the code 91D4₁₆, SUBX is eliminated as a possibility. The decomposition of the binary code is shown in Figure 3-26. ■

- We'll work this in class:

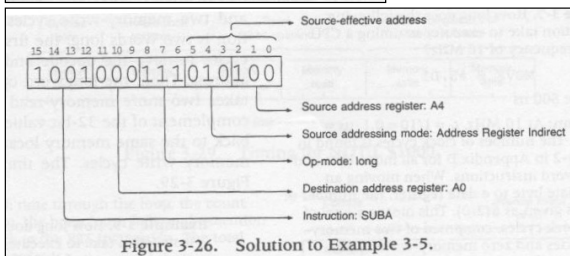


Figure 3-26. Solution to Example 3-5.



M68K Instruction Encoding – Example 3-6

Source: MacKenzie © 1995 Prentice Hall Publishers

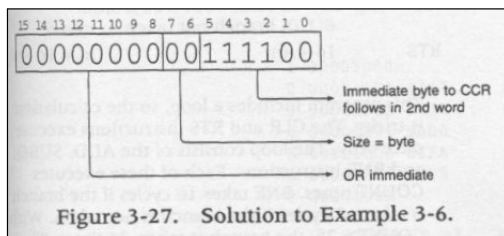
Example 3-6. Convert the 68000 machine-language instruction $003C_{16}$ to assembly language.

Solution:

```
ORI #data,CCR
```

Discussion: The first four bits of the operation, 0000_2 , delimit the possibilities to instructions on pages 366–368 in Appendix C. Again, a process of elimination is used to narrow in on the correct instruction. For example, AND immediate has bit 9 = 1. Since, bit 9 = 0 in the code $003C_{16}$, AND immediate is eliminated. The decomposition of the binary code is shown in Figure 3-27. Since the example only provides the first instruction word, the immediate data cannot be specified in the assembly-language form. ■

- We'll work this in class:



© 2002 Dr. James P. Davis Page 11

M68K Instruction Timing – Discussion

Source: MacKenzie © 1995 Prentice Hall Publishers

Each instruction on the 68000 takes a specific amount of time to execute.² The execution time is related to two parameters:

n the number of CPU clock cycles in an instruction
 t_c the period of the CPU clock

- When writing assembly code, it is still an objective to make programs either fast or “tight”.
- Writing fast code means we need to look at how long each instruction takes to execute (as CPU clock cycles)
- We will want to select code style (and specific instruction sequences) that take least number of CPU cycles.
- This affects program design, since there are many ways to code up an algorithm.



© 2002 Dr. James P. Davis Page 12

M68K Instruction Timing – Example 3-7

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-7. How long does the following instruction take to execute, assuming a CPU clock frequency of 10 MHz?

```
MOVE.B #5, D1
```

Solution: 800 ns

Discussion: At 10 MHz, $t_c = 1/10 = 0.1 \mu s = 100 \text{ ns}$. The number of clock cycles is found in Table D-2 in Appendix D for all move byte and move word instructions. When moving an immediate byte to a data register, the number of cycles is given as $8(2/0)$. This means a total of eight clock cycles, composed of two memory-read cycles and zero memory-write cycles. Therefore, the execution time is $8 \times 100 = 800 \text{ ns}$. ■

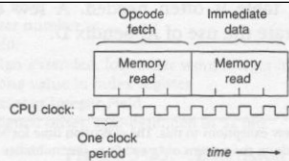


Figure 3-28. Timing for MOVE.B #data, Dn.

- We'll work this in class:



© 2002 Dr. James P. Davis Page 13

M68K Instruction Timing – Example 3-8

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-8. If a 68000 is operating from a 16-MHz clock, how long does the following instruction take to execute?

```
NOT.L $1000
```

Solution: 1.5 μs

Discussion: At 16 MHz, $t_c = 1/16 = 0.0625 \mu s = 62.5 \text{ ns}$. The instruction logically complements a longword in a memory location. From Table D-6 in Appendix D, the number of CPU cycles is given as $12(1/2)+$. The number of cycles is 12 (1 read, 2 writes) plus additional cycles for the addressing mode to specify the memory location. The addressing mode is absolute short; so, from Table D-1 an additional 12 cycles are added, for a total of 24 cycles. The execution time, therefore, is $24 \times 0.0625 = 1.5 \mu s$. ■

- We'll work this in class:



© 2002 Dr. James P. Davis Page 14

M68K Instruction Timing – Example 3-9

Source: MacKenzie © 1995 Prentice Hall Publishers

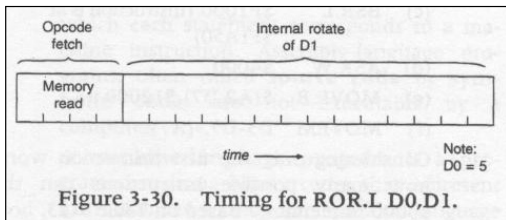
Example 3-9. How long does the following instruction take to execute, assuming a CPU clock frequency of 10 MHz and a value of 5 in D0?

```
ROR.L D0, D1
```

Solution: 1.8 μ s

Discussion: This instruction rotates the 32-bit contents of D1 right five positions. The number of CPU clock cycles is given in Table D-7 in Appendix D as $8 + 2n(1/0)$, where n is the number of positions to rotate. Only one memory-read cycle is required. The rest of the clock cycles are for the internal execution of the instruction. With $n = 5$, the total number of clock cycles is $8 + 2 \times 5 = 18$. At a clock frequency of 10 MHz, $t_c = 0.1 \mu$ s, so the execution time is $0.1 \times 18 = 1.8 \mu$ s. ■

- We'll work this in class:



© 2002 Dr. James P. Davis Page 15

M68K Instruction Timing – Example 3-10

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-10. The subroutine in Figure 3-31 adds a list of 16-bit values and returns the sum in D7. Parameters are passed to the subroutine as follows: The address of the list is in A0, the count is in D0. Assuming a count of 25₁₀, how long will this subroutine take to execute on a 10 MHz 68000?

Solution: 56.8 μ s

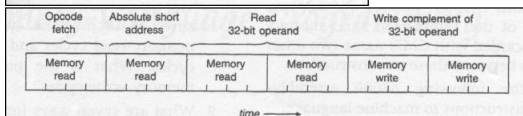
Discussion: The number of cycles for each instruction is given in Appendix D. These are summarized below.

CLR.W	4(1/0)	$n_1 = 4$
ADD.W	4(1/0) + 4(1/0)	$n_2 = 8$
SUBQ.B	4(1/0)	$n_3 = 4$
BNE.S	10(2/0) branch taken 8(1/0) branch not taken	$n_4 = 10$ $n_5 = 8$
RTS	16(4/0)	$n_6 = 16$

$$\begin{aligned}
 n &= n_1 + \text{COUNT}(n_2 + n_3) + (\text{COUNT} - 1)n_4 \\
 &\quad + n_5 + n_6 \\
 &= 4 + 25(8 + 4) + 24(10) + 8 + 16 \\
 &= 568
 \end{aligned}$$

At 10 MHz, the subroutine takes $0.1 \times 568 = 56.8 \mu$ s to execute. ■

- We'll work this in class:



```

1 00001000                ORG        $1000
2 00001000 4247          ADDLIST  CLR.W    D7
3 00001002 DE58          LOOP     ADD.W    (A0)+, D7
4 00001004 5300          SUBQ.B  #1, D0
5 00001006 66FA          BNE.S   LOOP
6 00001008 4E75          RTS
7 0000100A                END
    
```

Figure 3-31. ADDLIST subroutine.



© 2002 Dr. James P. Davis Page 16

M68K Instruction Timing – Example 3-11

Source: MacKenzie © 1995 Prentice Hall Publishers

Example 3-11. The RAM interface on a 10 MHz 68000 system inserts one wait state for each read or write cycle. How long will the following instruction take to execute on this system?

```
BSR, S    COSINE
```

Solution: 2.2 μ s

Discussion: From Table D-9 in Appendix D, the BSR instruction takes 18(2/2) cycles to execute. Since there are four memory references (2 read cycles, 2 write cycles), four extra cycles are required for the wait states, for a total of $18 + 4 = 22$ cycles. At 10 MHz, $t_c = 100$ ns, so the execution time is $22 \times 100 = 2200$ ns = 2.2 μ s. ■

- We'll work this in class:

Table D-9. Conditional and Branch Instruction Execution Times.

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10(2/0)	8(1/0)
	Long	10(2/0)	12(2/0)
BRA	Byte	10(2/0)	-
	Long	10(2/0)	-
BSR	Byte	18(2/2)	-
	Long	18(2/2)	-
DBcc	cc True	-	12(2/0)
	cc false, Count Not Expired	10(2/0)	-
	cc false, Count Expired	-	14(3/0)

