CONTROL PROGRAM INSTRUCTION SET

- Unconditional Branch
- Conditional Branch
- Choosing Instruction
- Repeat Instruction

Jump and Unconditional Branch

• The JMP and BRA instruction always change the flow of program.

• BRA instruction use the relative mode so it can only jump to -128 till 127 instruction.

• JMP able to jump into anywhere in the 64k address space.

•BRA * will stop the CPU movement with execution of this instruction repeatedly.

• BRN is similar with the NOP function which do 3 cycle, and ease the design of compiler.

Mnemonic	Function	DIR	EXT	INDX	INDY	INH	REL
JMP	Jump	X	X	X	X		
BRA	Branch Always						X
BRN	Branch Never						X

Repeating the instructions

```
*
* Input from PORTC, invert, output to PORTB
* Do repeatedly
*
PORTC     EQU   4
PORTB     EQU   3
REGS      EQU   $1000

          LDX   #REGS
REPEAT    LDAA  PORTC, X
          COMA
          STAA  PORTB, X
          BRA   REPEAT
```


Conditional Branch

- Branch is easy with using NZVC.
- Normally follow CBA, CMP, CPX, CPY, SBA, or SUB instruction.
- For zero value or negative, can directly follow the LDA, TST, and BIT instruction.

Mnemonic	Function	IMM	DIR	EXT	INDX	INDY	REL	Condition
BCC	Branch if Carry Clear						X	C = 0
BCS	Branch if Carry Set						X	C = 1
BEQ	Branch if Equal Zero						X	Z = 1
BMI	Branch if Minus						X	N = 1
BNE	Branch if Not Equal Zero						X	Z = 0
BPL	Branch if Plus						X	N = 0
BVC	Branch if Overflow Clear						X	V = 0
BVS	Branch if Overflow Set						X	V = 1

Simple Conditional Branch

```
* Negate ACCA if it content negative value
*
        TSTA
        BPL     NEXT
        NEGA
NEXT    …

* Send character 'A' to PORTB if value PORTC = 0
*
        TST     $1003   ;PORTC = $1003
        BNE     SKIP
        LDAA    #'A
        STAA    $1004   ;PORTB = $1004
SKIP    …
```


Simple Conditional Branch

```
*
* Send character 'A' to PORTB if A + B produce carry
* Send 'B' for others value
*
                ABA
                BCS     HAVECARRY
                LDAA    #'B
                BRA     SEND
HAVECARRY       LDAA    #'A
SEND            STAA    $1004   ;PORTB = $1004
```

Signum

```
*
* If A positive, replace with 1
* If A negative, replace with -1
* If A zero, let it be
*
                TSTA
                BEQ     DONE
                BMI     NEGATIVE
                LDAA    #1
                BRA     DONE
NEGATIVE        LDAA    #-1
DONE            …
```


Simple Conditional Branch

```
*
* Loop to total up $E000… $E00F
*
ARRAY       EQU     $E000

        LDAA    #16     Timer = #element (16 byte overall)
        CLRB            clear accumulator
        LDY     #ARRAY      set pointer to the early array
LOOP    ADDB    0,Y     add current element
        INY             amortize pointer
        DECA            depreciate timer
        BNE     LOOP    repeat till finish
```

Simple Conditional Branch

```
*
* Loop to delay the CPU
* Delay = tLDAB  + (tDECB + tBNE) x count
*
[2]              LDAB   #COUNT      load B with count
[2]  DELAY       DECB                Decrement B
[2]              BNE    DELAY       If B≠0, goto Delay
```

Delay minimum when COUNT = 1:
DELAY = 2 + (2 + 2) * 1 = 6 clock cycle
Delay maximum when COUNT = 0 (why?) :
DELAY = 2 + (2 + 2) * 256 = 1026 clock cycle

Simple Conditional Branch

- BHS instruction is same as effect with BCC.
- BLO instruction have similar effect with BCS.
- Use the unsigned data for data that possible to be negative like:
  - > rainfall
  - > wind speed
  - > moist

Mnemonic	Function	IMM	DIR	EXT	INDX	INDY	REL	Condition
BHI	Branch if Higher						X	Unsigned > (C+Z = 1)
BHS	Branch if Higher or Same						X	Unsigned ≥ (C = 0)
BLO	Branch if Lower						X	Unsigned < (C = 1)
BLS	Branch if Lower or Same						X	Unsigned ≤ (C + Z = 0)

Branch Unsigned Number

```
* Send character 'A' to PORTB if the value PORTC > 25
* Send 'B' for others value
*
PORTC  EQU    4
PORTB  EQU    3
REGS   EQU    $1000

       LDAA   PORTC, X
       COMA   #25
       BGT    OUT_A
       LDAA   #'B
       BRA    OUT
OUT_A  LDAA   #'A
OUT    STAA   PORTB, X
```


Branch Signed Number

- Use the signed data for data that possible negative like:
  - ~ temperature
  - ~ voltage
  - ~ Current
  - ~ micromouse coordinate when comparing a second before

Mnemonic	Function	IMM	DIR	EXT	INDX	INDY	REL	Condition
BGE	Branch if Greater Than or Equal						X	Signed ≥ (N ⊕ V=0)
BGT	Branch if Greater Than						X	Signed > (Z · (N ⊕ V = 0)
BLE	Branch if Less Than of Equal						X	Signed ≤ (Z · (N ⊕ V = 1))
BLT	Branch if Less Than						X	Signed < (N ⊕ V=1)

Branch Signed Number

* Send $0F to PORTB if ACCA < -20
* Send $0F if ACCA > +30
* Send $00 for others value

```
                CMPA  #-20
                BLT   COLD
                CMPA  #30
                BGT   HOT
DEFAULT         CLRB
                BRA   DONE
HOT             LDAB  #$0F
DONE            STAB  $1004
```


Test Data & Bit Expression Instruction

- BITA, BITB do the operation ANDA and ANDB, update N and Z without changing the operand.
- BRCLR, BRSET: branch if certain bits in memory, 0 or zero.
- BCLR & BSET change the memory without using ACCA/B, with one instruction only.

Mnemonic	Function	Operation	IMM	DIR	EXT	INDX	INDY	INH
BITA	Test bits in A	A ^ M		X	X	X	X	
BITB	Test bits in B	B ^ M						
BRCLR	Branch if bits clear	? M ^ mm = 0		X		X	X	
BRSET	Branch if bits set	? M ^ mm = 1		X		X	X	

Test Data & Bit Expression Instruction

* The example of using the "spin loop":
* Wait MSB in PORTC = 1 before read PORTE

```
            LDX   #REGS
            LDAA  #%10000000
SPIN        BITA  PORTC, X
            BEQ   SPIN
            LDAA  PORTE, X
```

* The same segment using BCLR

```
            LDX   #REGS
SPIN        BCLR  PORTC, X    %10000000    SPIN
            LDAA  PORTE, X
```


Counting Bit

```
* Count total of 1 in ACCA
*
            CLRB          ; count = 0
REPEAT      TSTA          ; if ACCA = 0 then quit
            BEQ   DONE
            LSLA
            BCC   REPEAT
            INCB
            BRA   REPEAT
DONE …
```

If any door is opened, sound the alarm

```
*
* PC0 = 1 when DOOR1 open, PC1 = 1 when DOOR2 open
* PB0 = 1 sound the alarm
* Example 1: using the BITA + BEQ instruction
*
              LDX   #REGS
              LDAA  #%00000011    load mask
LOOP          BITA  PORTC, X
              BEQ   SECURE
ALARM         BCLR  PORTB, X  %00000001
              BRA   LOOP
SECURE              BSET   PORTB, X  %00000001
              BRA   LOOP
```


If any door is opened, sound the alarm

```
*
* PC0 = 1 when DOOR1 open, PC1 = 1 when DOOR2 open
* PB0 = 1 sound the alarm
* Example 2: using the BCLR instruction
*
              LDX   #REGS
LOOP          BCLR  PORTC, X  %00000011 SECURE
ALARM         BCLR  PORTB, X  %00000001
              BRA   LOOP
SECURE        BSET  PORTB, X  %00000001
              BRA   LOOP
```

If any door is opened, sound the alarm

```
*
* PC0 = 1 when DOOR1 open, PC1 = 1 when DOOR2 open
* PB0 = 1 sound the alarm
* Example 3: using the BRSET instruction
*
            LDX    #REGS
LOOP        BRSET  PORTC, X  %00000001 ALARM
            BRSET  PORTC, X  %00000010 ALARM
SECURE      BSET   PORTB, X   %00000001
            BRA    LOOP
ALARM       BCLR   PORTB, X  %00000001
            BRA    LOOP
```


Multiple Branch

Useful for jump to some different segment depends on the received value

```
switch(i) {
  case 0: do action 0; break;
  case 1: do action 1; break;
  ...
  case n: do action n; break;
  default: exception
}
```

Multiple Branch using IF case

```
*
*Value B determine the segment (ACTION0-ACTIONN) that to
be generated. N = valid value for maxB
*
        TSTB
        BEQ     ACTION0
        CMPB    #1
        BEQ     ACTION1
        CMPB    #2
        BEQ     ACTION2
        …
        CMPB    #N
        BEQ     ACTIONN
        default action
```


Multiple Branch using JMP Table

```
*
*Value B determine the segment (ACTION0-ACTIONN) that to
be generated.  Max = N = valid value for maxB
*
        LDX     #TABLE
        CMPB    #MAX            check B <= MAX
        BLE     INRANGE
        default action
INRANGE ABX                     X = X + B
        ASLB                    B = 2B
        ABX                     X = X + 2B
        JMP     0,X
TABLE   BRA     ACTION0
        BRA     ACTION1
        ...
        BRA     ACTIONN
```

from case jump table to true code

Program Execution Time

A easy way to create a delay is to use program loops. Use the instructions in the table as an example.

Execution Times of a Sample of Instructions

Instruction	Execution Time (E-clock Cycles)
BNE <rel>	3
DECB	2
DEX	3
LDAB <imme>	2
LDX <imme>	3
NOP	2

The following instruction sequence takes 5 μs to execute for 2 MHz E-clock signal.

```
again   nop             ; 2 E cycles
        nop             ; 2 E cycles
        dex             ; 3 E cycles
        bne again       ; 3 E cycles
```


Example: Write a program loop to create a delay of 100 ms.
Solution: A delay of 100 ms can be created by repeating the previous loop 20000 times

The following instruction sequence creates a delay of 100 ms.

```
        ldx   #20000
again   nop
        nop
        dex
        bne   again
```

Longer delays can be created by using nested program loops.

Example: Write a program to create a 10-second delay.
Solution: A 10-second delay can be created by repeating the loop in the example above 100 times.

```
        ldab  #100
outer   ldx   #20000
inner   nop
        nop
        dex
        bne   inner
        decb
        bne   outer
```

System Control Instruction

- ✓ NOP – no operation held the 2 cycle delay.
- ✓ STOP – if bit S in CCR = 0 then stop the clock to reduce the power usage. If S = 1 is equals to NOP.
- ✓ TEST – test instruction used by the manufacturer.

Mnemonic	Function	INH
NOP	No Operation	X
STOP	Stop Clocks	X
TEST	Special Test Mode	X