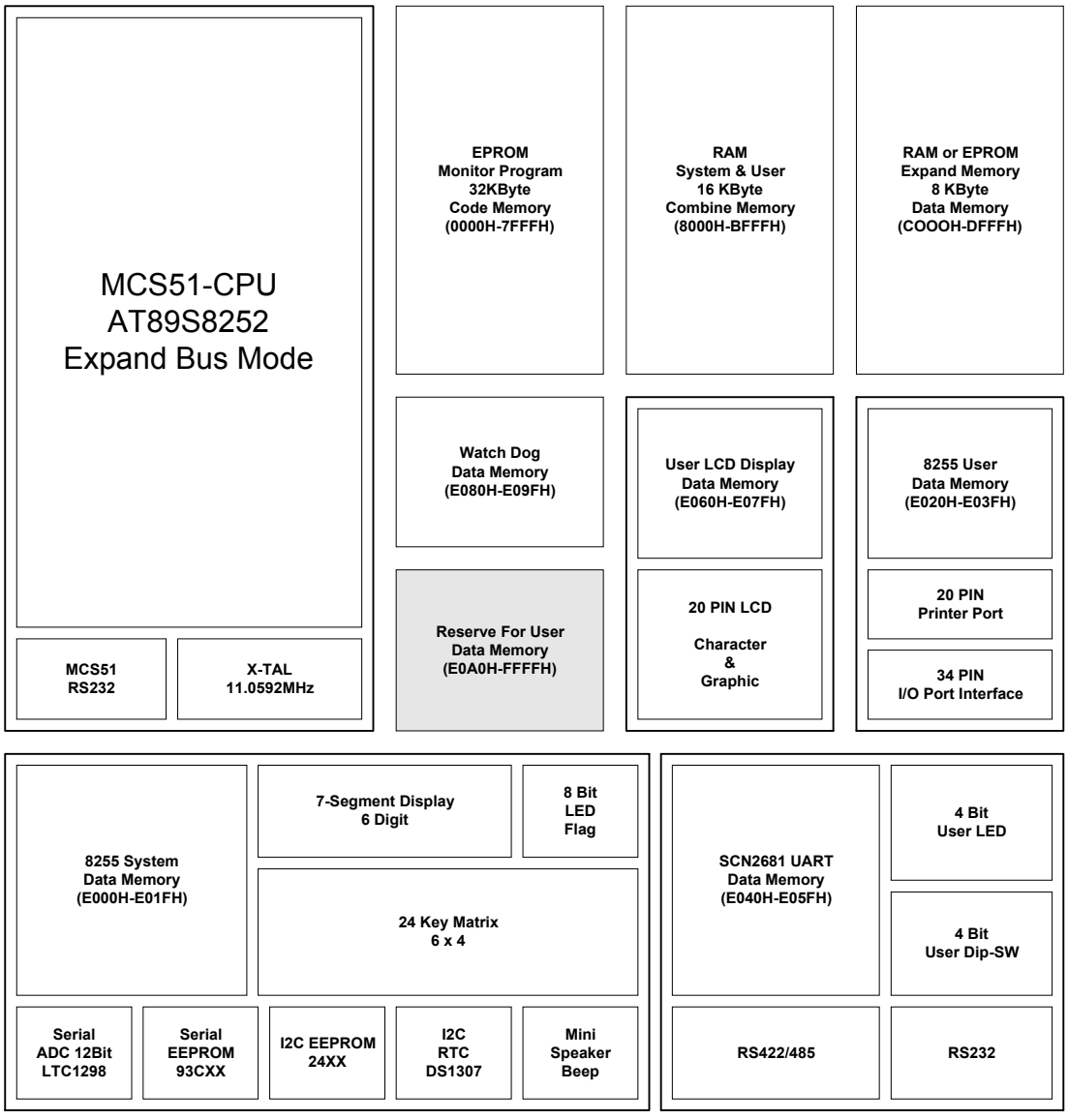
โครงสร้างทางฮาร์ดแวร์ของ MCS51 Single Board

เมื่อผู้ใช้งานกำหนดโหมดการทำงานของ ET-BOARD V6.0 ให้เป็น MCS51 Mode แล้วนั้น ระบบบัส ระบบหน่วยความจำ และอุปกรณ์ I/O ต่างๆ ภายในบอร์ด จะถูกควบคุมด้วย MCS51 ทั้งหมด ส่วน Z80 นั้น จะถูกตัดการทำงานออกจากระบบไปทั้งหมด ซึ่งในโหมด MCS51 นี้ จะมีอุปกรณ์มากกว่าของ Z80 อยู่หลายส่วน ซึ่งเป็นส่วนที่บรรจุมากับตัว MCS51 เองทั้งสิ้น โดยอุปกรณ์เหล่านี้มีทั้งที่มีการเชื่อมต่อออกมาใช้งานภายนอก เช่น พอร์ตอนุกรม RS232 ช่องที่เป็นของ MCS51 เอง นอกจากนี้แล้วยังมีส่วนของอุปกรณ์ที่บรรจุไว้ภายในตัวของ CPU เอง เช่น Timer/Counter และหน่วยความจำ EEPROM เป็นต้น โดยโครงสร้างของ ET-BOARD V6.0 ในโหมด MCS51 นั้น ในส่วนที่มีการออกแบบให้เชื่อมต่อกับอุปกรณ์ภายนอกไว้มีดังนี้



Block Diagram ของ ET-BOARD V6.0 ในโหมด MCS51 Single Board

การจัดสรรหน่วยความจำของ MCS51 Single Board

ตามปรกติแล้ว MCS51 มาตรฐาน จะสามารถอ้างถึงหน่วยความจำได้ทั้งหมด 128Kbyte โดยแบ่งออกเป็นส่วนของหน่วยความจำสำหรับเก็บโปรแกรม (Code Memory หรือ Program Memory) จำนวน 64Kbyte และส่วนของหน่วยความจำสำหรับเก็บข้อมูล (Data Memory) อีก 64Kbyte โดยหน่วยความจำทั้ง 2 ส่วน นี้จะมีตำแหน่งแอดเดรสในการเข้าถึงที่เหมือนกัน คือ 0000H-FFFFH แต่วิธีการเข้าถึงหน่วยความจำ ทั้ง 2 ส่วน นี้จะแยกออกจากกัน อย่างชัดเจน ด้วยรูปแบบของคำสั่ง และสัญญาณในการเข้าถึงหน่วยความจำ โดยในส่วนของ ET-BOARD V6.0 นั้น จะออกแบบระบบหน่วยความจำของ MCS51 ให้แยกเป็น 4 ส่วน ดังนี้

1. **Monitor Program** ซึ่งเป็นส่วนของหน่วยความจำสำหรับเก็บโปรแกรม Monitor ของบอร์ด โดยใช้อุปกรณ์หน่วยความจำเป็น EPROM เบอร์ 27C010 ซึ่งตามปรกติจะมีขนาด 128Kbyte แต่มีการถอดรหัสตำแหน่งการทำงานของหน่วยความจำเพื่อใช้งานใน MCS51 Single Board ไว้จำนวน 32 KByte และใช้วิธีการเข้าถึงหน่วยความจำแบบ Code Memory
2. **User Memory (U4)** ซึ่งเป็นหน่วยความจำที่จัดสรรไว้สำหรับผู้ใช้ เพื่อใช้เขียนโปรแกรม หรือ เก็บข้อมูลก็ได้ตามต้องการ โดยส่วนนี้จะเป็น RAM เบอร์ 62256 ซึ่งตามปรกติจะมีขนาด 32Kbyte แต่มีการถอดรหัสตำแหน่งหน่วยความจำเพื่อใช้งานใน MCS51 Single Board จำนวน 16Kbyte โดยจะใช้การเข้าถึงหน่วยความจำแบบ Combine ซึ่งหมายความว่า หน่วยความจำส่วนนี้สามารถเข้าถึงได้ทั้งแบบ Code Memory หรือ Data Memory ก็ได้
3. **Data Memory (U5)** ซึ่งเป็นส่วนของหน่วยความจำ Expansion Memory สำหรับใช้เก็บข้อมูลเพียงอย่างเดียว แต่อุปกรณ์ของหน่วยความจำส่วนนี้สามารถเลือกใช้เป็น RAM เบอร์ 62256 หรือ EPROM เบอร์ 27256 ก็ได้ โดยมีการถอดรหัสตำแหน่งของหน่วยความจำส่วนนี้เพื่อใช้งานจำนวน 8 Kbyte และใช้วิธีการเข้าถึงหน่วยความจำแบบ Data Memory
4. **Port I/O** เป็นส่วนของหน่วยความจำ ซึ่งถูกจัดสรร ตำแหน่งแอดเดรสส่วนนี้ออกมาเพื่อใช้ติดต่อกับอุปกรณ์ I/O ภายนอก โดยใช้วิธีการแบบ Memory MAP I/O ซึ่งมีขนาด 8 Kbyte และใช้วิธีการเข้าถึงแบบ Data Memory

*****หมายเหตุ*****

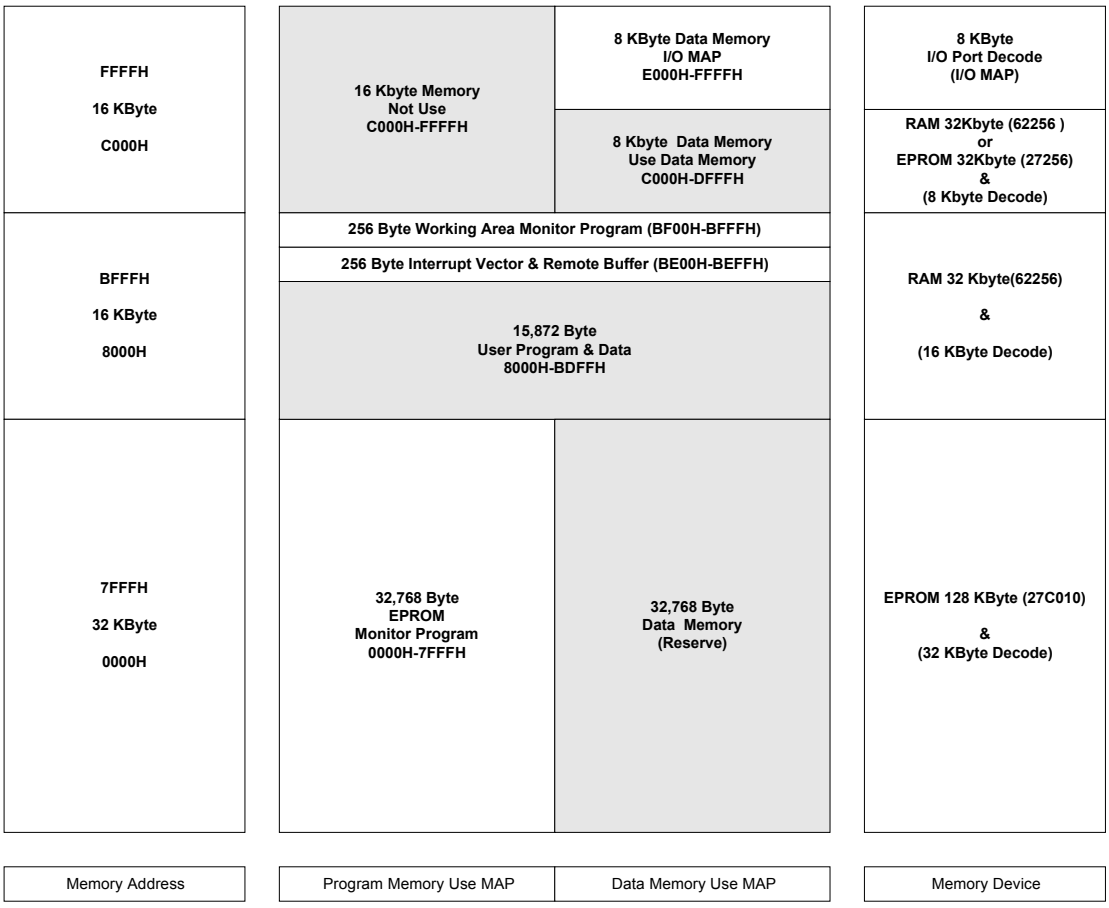
- **Program Memory หรือ Code Memory หมายถึง** พื้นที่ของหน่วยความจำสำหรับใช้เก็บรหัสคำสั่งของโปรแกรม ซึ่งสามารถอ่านออกมาใช้งานได้อย่างเดียว และใช้การเข้าถึงหน่วยความจำส่วนนี้ด้วยคำสั่ง MOVC และใช้สัญญาณ PSEN เป็นสัญญาณในการเข้าถึงหน่วยความจำ เช่น

```
MOV   DPTR,#7000H
CLR   A
MOVC  A,@A+DPTR          ; อ่านค่าจากตำแหน่ง 7000H
```

- **Data Memory หมายถึง** พื้นที่ของหน่วยความจำที่ใช้สำหรับเก็บข้อมูล ซึ่งสามารถอ่านและเขียนได้ แต่ไม่สามารถสั่ง Run โปรแกรมในพื้นที่ของหน่วยความจำส่วนนี้ได้ ซึ่งการเข้าถึงหน่วยความจำส่วนนี้จะใช้คำสั่ง MOVX และใช้สัญญาณ RD และ WR ในการเข้าถึงหน่วยความจำ เช่น

```
MOV     DPTR,#9000H
MOVX    A,@DPTR              ; อ่านข้อมูลจากตำแหน่ง 9000H
MOV     DPTR,#9500H
MOV     A,#31H
MOVX    @DPTR,A              ; เขียนค่า 31H ไปยังตำแหน่ง 9500H
```

- **Combine Memory หมายถึง** พื้นที่ของหน่วยความจำ ซึ่งสามารถใช้วิธีการเข้าถึงได้ทั้งแบบ Code Memory หรือ Data Memory ก็ได้ ดังนั้นพื้นที่ของหน่วยความจำส่วนนี้จึงสามารถใช้ประโยชน์ได้ ทั้งการเก็บข้อมูล และการเก็บคำสั่งของโปรแกรม รวมทั้งสามารถสั่ง Run โปรแกรมที่เก็บไว้ในพื้นที่ของหน่วยความจำส่วนนี้ได้อีกด้วย โดยวิธีการออกแบบวงจรของหน่วยความจำให้เป็นแบบ Combine Memory นั้น จะใช้วิธีการนำสัญญาณ PSEN และ RD มา AND กัน เพื่อนำสัญญาณไปเปิดข้อมูลจากหน่วยความจำ ซึ่งไม่ว่าจะให้คำสั่ง MOVX (ขาสัญญาณ RD Active) หรือใช้คำสั่ง MOVC(ขาสัญญาณ PSEN Active) ก็จะสามารถเข้าถึงข้อมูลในหน่วยความจำส่วนนี้ได้เหมือนกัน เพียงแต่มีข้อแม้ว่า วงจรถอดรหัสตำแหน่งแอดเดรสของหน่วยความจำแบบ Combine นี้ต้องมีตำแหน่งเหมือนกันทั้ง Code Memory และ Data Memory

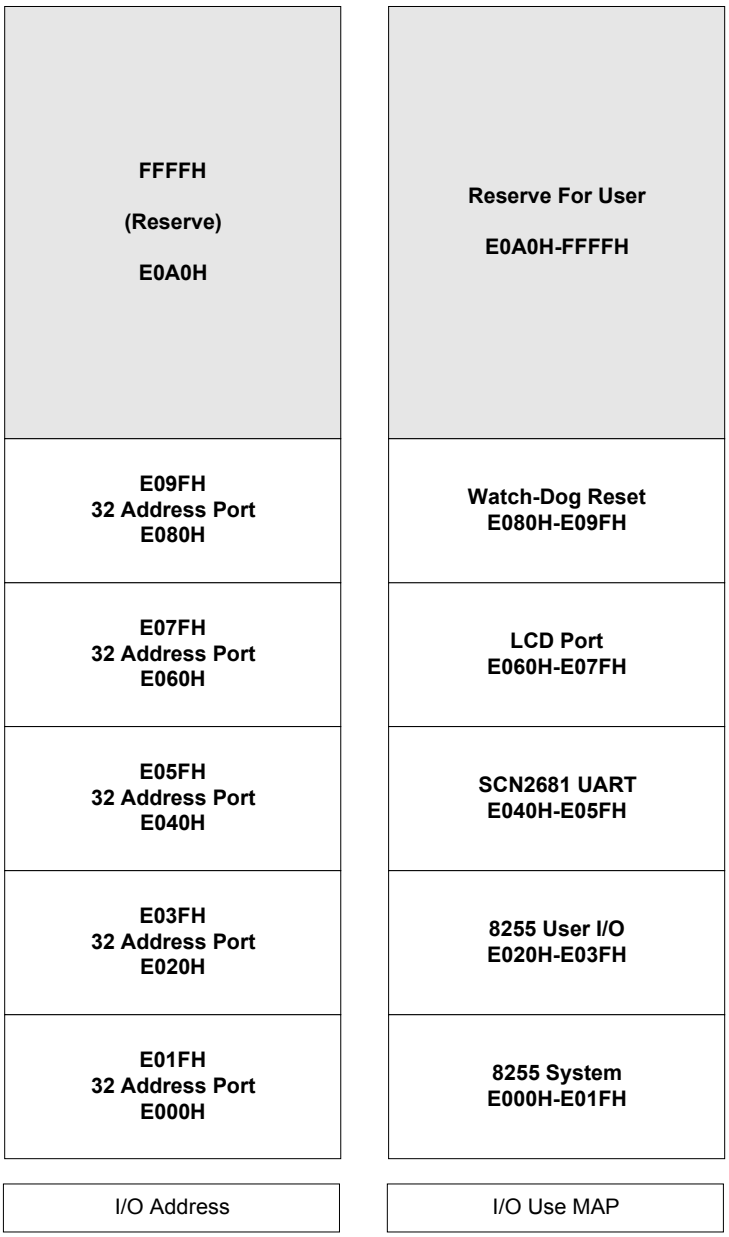


การจัดสรรหน่วยความจำ (Memory MAP) ของ ET-BOARD V6.0 ในโหมด MCS51 Single Board

การจัดสรรตำแหน่ง I/O Port ของ MCS51 Single Board

ตามปรกติแล้ว MCS51 ไม่มีสัญญาณสำหรับติดต่อกับ I/O ภายนอกโดยตรง ทั้งนี้เนื่องจาก MCS51 มีส่วนของ Port I/O รวมอยู่ด้วยแล้วบางส่วน เช่น Port P1 แต่อย่างไรก็ตามเนื่องจาก MCS51 สามารถติดต่อกับหน่วยความจำ Data Memory ภายนอกได้ ดังนั้น ET-BOARD V6.0 จึงได้จัดสรรตำแหน่งแอดเดรสของหน่วยความจำส่วนนี้จำนวน 8Kbyte เพื่อใช้ติดต่อกับส่วนของ I/O Port ภายนอก ตั้งแต่ E000H-FFFFH ซึ่งการเข้าถึงอุปกรณ์ I/O นั้น ก็จะใช้วิธีการเดียวกันกับการเข้าถึงหน่วยความจำข้อมูลทุกประการ เพียงแต่ผู้ใช้ต้องเข้าใจและแบ่งแยกให้ถูกว่า ช่วงตำแหน่งแอดเดรสใด ถูกจัดสรรให้เป็นของหน่วยความจำ ช่วงแอดเดรสใดถูกจัดสรรให้เป็นของ I/O Port เท่านั้นเอง

สำหรับอุปกรณ์ I/O Port ที่มีการออกแบบและจัดเตรียมไว้ให้ผู้ใช้งาน แล้วภายใน ET-BOARD V6.0 นั้นจะมีดังนี้



การจัดสรร I/O Port (I/O MAP) ของ ET-BOARD V6.0 ในโหมด MCS51 Single Board

การใช้งาน Single Board ในโหมด MCS51

ในโหมดการทำงานของ MCS51 นั้น สามารถเลือกรูปแบบการพัฒนาโปรแกรมของบอร์ดได้ 3 แนวทาง ซึ่งพอสรุปคุณสมบัติของวิธีการต่างๆให้เห็น ได้ดังนี้

1. ใช้การป้อนโปรแกรมเข้า Single Board โดยตรง ซึ่งวิธีการนี้ไม่จำเป็นต้องเชื่อมต่อกับคอมพิวเตอร์ PC สามารถศึกษาทดลองโดยใช้ Single Board เพียงอย่างเดียวได้ แต่วิธีการนี้เหมาะสำหรับการศึกษาโปรแกรมที่มีขนาดเล็กๆ เนื่องจากต้องใช้วิธีการป้อนรหัสคำสั่งของโปรแกรม ให้กับ Single Board โดยตรง ซึ่งถ้าโปรแกรมมีความยาวมากๆจะทำให้เสียเวลาในการป้อนโปรแกรมมาก และเมื่อมีการแก้ไขการทำงานของโปรแกรมโดยการเพิ่มเติมหรือลดคำสั่งแต่ละครั้งก็จำเป็นต้องป้อนรหัสคำสั่งโปรแกรมใหม่ด้วย

2. ใช้วิธีการพัฒนาโปรแกรมด้วย REMOTE-32 ซึ่งวิธีการนี้ ต้องทำการเชื่อมต่อ Single Board กับคอมพิวเตอร์ PC ทางพอร์ตอนุกรม RS232 ด้วย ซึ่งวิธีการนี้จะสะดวกมาก เนื่องจากสามารถเขียน Source Code โปรแกรมเก็บไว้ในคอมพิวเตอร์ PC และทำการบันทึกเป็นแฟ้มเก็บไว้ได้ และเมื่อต้องการแก้ไขหรือเปลี่ยนแปลงการทำงานของโปรแกรมก็สามารถแก้ไขหรือดัดแปลง Source Code แล้วทำการสั่งแปลโปรแกรมนั้นให้ได้เป็นรหัสคำสั่ง แบบ Intel HEX จากนั้นจึงสั่ง Download ข้อมูล Intel HEX นั้นลงมายังหน่วยความจำของ Single Board เพื่อสั่ง Run ได้

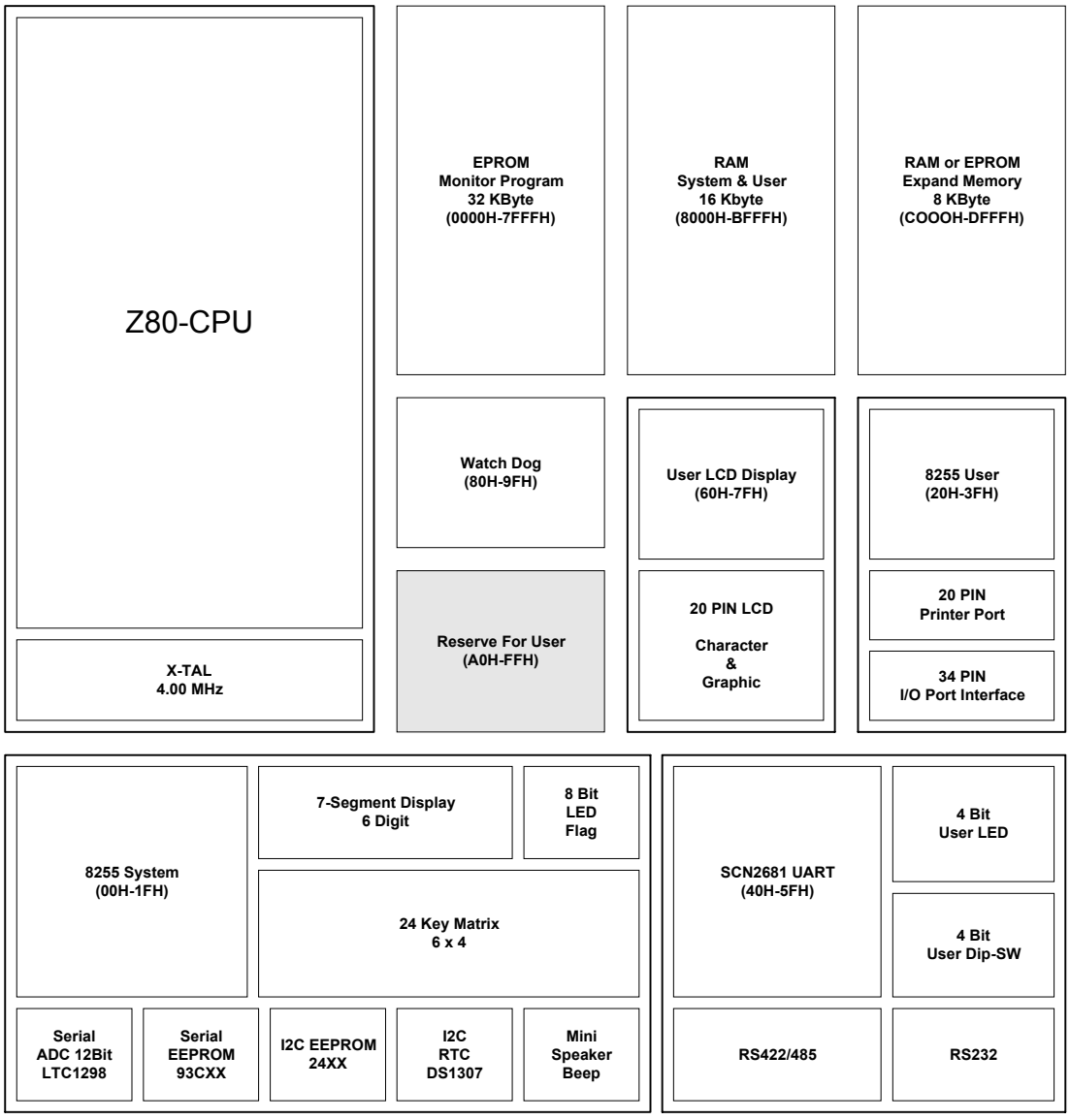
   โดยวิธีการนี้ต้องติดตั้งโปรแกรม Procomm Plus บนเครื่องคอมพิวเตอร์ PC เพื่อใช้ในการติดต่อสื่อสารระหว่าง Single Board กับ คอมพิวเตอร์ PC ด้วย นอกจากนี้แล้ว ยังต้องติดตั้งโปรแกรม Assembler ของ MCS51 เพื่อใช้สำหรับเขียนโปรแกรมและแปลโปรแกรมให้เป็นรหัสคำสั่งแบบ Intel HEX เพื่อใช้ Download ให้กับหน่วยความจำของ Single Board เพื่อสั่ง Run ในภายหลังด้วย

3. ใช้วิธีการพัฒนาโปรแกรมด้วย EMULATE-8051 ซึ่งวิธีการนี้เหมาะสำหรับผู้เริ่มต้นศึกษาโปรแกรมเป็นอย่างมาก แต่ก็จำเป็นต้องเชื่อมต่อ Single Board เข้ากับเครื่อง คอมพิวเตอร์ PC ทางพอร์ตอนุกรม RS232 ด้วย เช่นกัน ซึ่งวิธีการนี้ จะมีข้อดี คือสามารถ ศึกษาการทำงานของ CPU ได้อย่างละเอียดโดยใช้งานร่วมกับโปรแกรม Simulate บน คอมพิวเตอร์ PC ซึ่งผู้ใช้สามารถเห็นการทำงานของคำสั่งต่างๆบนหน้าจอคอมพิวเตอร์ ว่าเมื่อ CPU ปฏิบัติคำสั่งใดแล้วส่งผลเปลี่ยนแปลงต่อหน่วยความจำหรือรีจิสเตอร์ รวมทั้งอุปกรณ์ I/O อะไรบ้าง

   โดยวิธีการนี้ต้องติดตั้งโปรแกรม MONICA52 บนเครื่องคอมพิวเตอร์ PC เพื่อใช้ในการติดต่อสื่อสารระหว่าง Single Board กับ คอมพิวเตอร์ PC ด้วย นอกจากนี้แล้ว ยังต้องติดตั้งโปรแกรม Assembler ของ MCS51 เพื่อใช้สำหรับเขียนโปรแกรมและแปลโปรแกรมให้เป็นรหัสคำสั่งแบบ Intel HEX เพื่อใช้ Download ให้กับหน่วยความจำของ Single Board เพื่อสั่ง Run ในภายหลังด้วย

โครงสร้างทางฮาร์ดแวร์ของ Z80 Single Board

เมื่อผู้ใช้งานกำหนดโหมดการทำงานของ ET-BOARD V6.0 ให้เป็น Z80 Mode แล้วนั้น ระบบบัส ระบบหน่วยความจำ และ อุปกรณ์ I/O ต่างๆ ภายในบอร์ดจะถูกควบคุมด้วย Z80 ทั้งหมด ส่วน MCS51 นั้นจะถูกตัดการทำงานออกจากระบบไปทั้งหมด ซึ่งในโหมด Z80 นี้ จะมีอุปกรณ์เพียงชุดเดียวของ ET-BOARD V6.0 ที่ Z80 ไม่สามารถนำมาใช้งานได้ คือ พอร์ตสื่อสาร RS232 ช่องที่เป็นของ MCS51 เท่านั้น ส่วนพอร์ตสื่อสาร RS232 และ RS422/485 ที่เป็นของ Chips Support เบอร์ SCN2681 นั้น Z80 สามารถเข้าถึงและใช้งานได้ตามปรกติทุกอย่าง โดยโครงสร้างการทำงานของ ET-BOARD V6.0 ในโหมด Z80 นั้นจะเป็นดังนี้

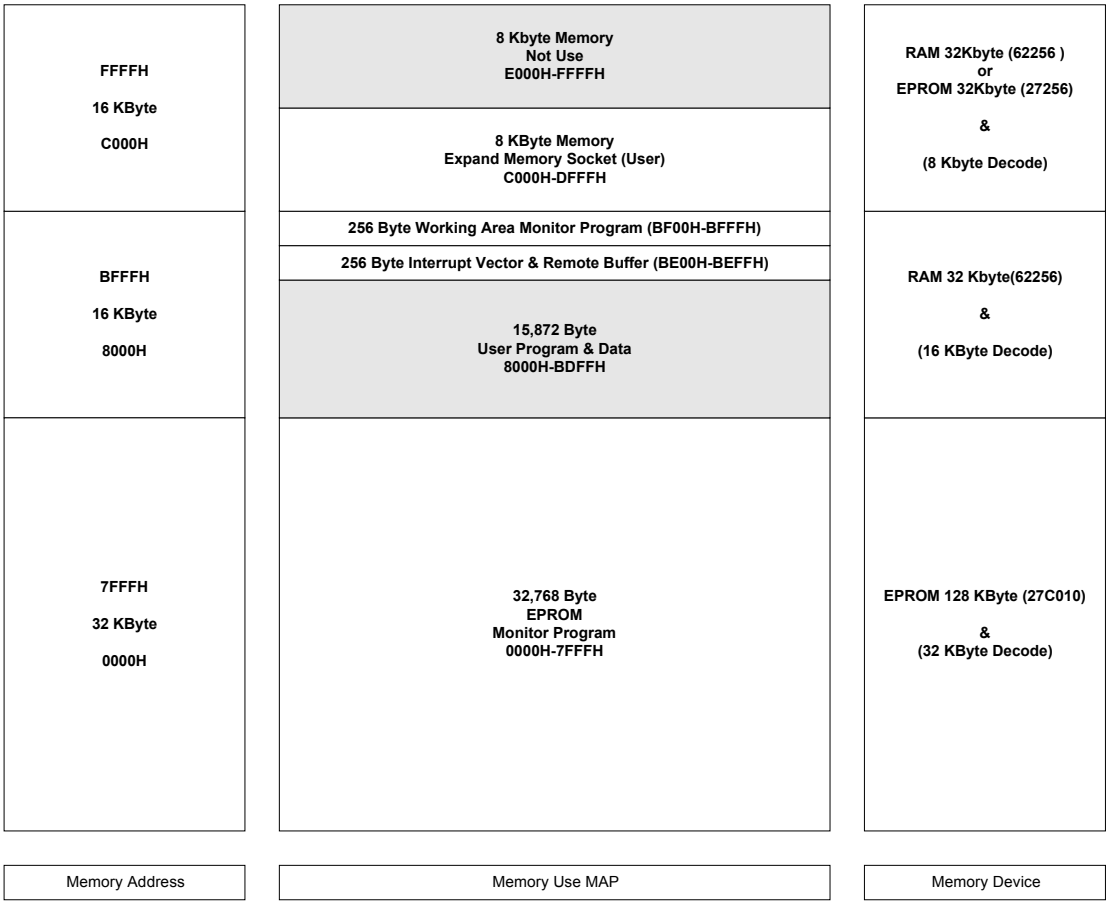


Block Diagram ของ ET-BOARD V6.0 ในโหมด Z80 Single Board

การจัดสรรหน่วยความจำของ Z80 Single Board

ตามปรกติแล้ว Z80 จะสามารถอ้างหน่วยความจำได้ทั้งหมด 64Kbyte โดยรวมทั้งส่วนที่เป็นของหน่วยความจำสำหรับเก็บข้อมูลและหน่วยความจำสำหรับเก็บโปรแกรมรวมไว้ด้วยกัน ดังนั้นจึงต้องมีการจัดสรรตำแหน่งแอดเดรสของหน่วยความจำทั้ง 64Kbyte นี้ ว่าจะให้ช่วงตำแหน่งแอดเดรสใดเป็นพื้นที่สำหรับเก็บโปรแกรมและตำแหน่งใดเป็นพื้นที่สำหรับเก็บข้อมูล

ซึ่งในส่วนของ ET-BOARD V6.0 นั้น เมื่อกำหนดโหมดการทำงานเป็น Z80 ไว้ การจัดสรรตำแหน่งของหน่วยความจำจะเป็นดังนี้



การจัดสรรหน่วยความจำ (Memory MAP) ของ ET-BOARD V6.0 ในโหมด Z80 Single Board

*****หมายเหตุ*****

- สำหรับหน่วยความจำ RAM ตำแหน่ง U4 (8000H-BFFFH) นั้น ในบอร์ดจะออกแบบให้ใช้เบอร์ 62256 ซึ่งมีขนาดของหน่วยความจำเป็น 32Kbyte ไว้ แต่ในการใช้งานจริงจะสามารถอ้างตำแหน่งแอดเดรสของ RAM นี้ได้ 16Kbyte (8000H-BFFFH) เท่านั้น
- ส่วนหน่วยความจำ U5 นั้น เป็นส่วนของ Expansion Memory ซึ่งสามารถติดตั้ง ได้ทั้ง EPROM และ RAM แต่ในการใช้งานจะสามารถอ้างตำแหน่งได้เพียง 8Kbyte(C000H-DFFFH) เท่านั้น

การจัดสรร I/O ของ Z80 Single Board

ตามปรกติแล้ว Z80 จะสามารถอ้างตำแหน่ง I/O พอร์ต เพื่อติดต่อกับอุปกรณ์ I/O ภายนอกได้ทั้งหมด 256 ตำแหน่ง Port แต่สำหรับ ET-BOARD V6.0 นั้นจะทำการ Decode ตำแหน่งของ I/O พอร์ตสำหรับใช้งานภายในบอร์ดไว้แล้ว จำนวน 160 ตำแหน่ง (00H-9FH) และว่างไว้ให้ผู้ใช้งานสามารถเพิ่มเติมได้เองอีกจำนวน 96 ตำแหน่ง โดยอุปกรณ์ I/O ต่างๆที่ออกแบบไว้แล้วภายในบอร์ดนั้น จะถูก Decode ตำแหน่งไว้อุปกรณ์ละ 32 ตำแหน่ง ดังนี้

I/O Address	I/O Use MAP
FFH **96 Address Port** **(Reserve)** **A0H**	**Reserve For User** **A0H-FFH**
9FH **32 Address Port** **80H**	**Watch-Dog Reset** **80H-9FH**
7FH **32 Address Port** **60H**	**LCD Port** **60H-7FH**
5FH **32 Address Port** **40H**	**SCN2681 UART** **40H-5FH**
3FH **32 Address Port** **20H**	**8255 User I/O** **20H-3FH**
1FH **32 Address Port** **00H**	**8255 System** **00H-1FH**

การจัดสรร I/O (I/O MAP) ของ ET-BOARD V6.0 ในโหมด Z80 Single Board

การใช้งาน Single Board ในโหมด Z80

ในโหมดการทำงานของ Z80 นั้น สามารถเลือกรูปแบบการพัฒนาโปรแกรมของบอร์ดได้ 2 แนวทาง ซึ่งพอสรุปคุณสมบัติของวิธีการต่างๆให้เห็น ได้ดังนี้

1. ใช้การป้อนโปรแกรมเข้า Single Board โดยตรง ซึ่งวิธีการนี้ไม่จำเป็นต้องเชื่อมต่อกับคอมพิวเตอร์ PC สามารถศึกษาทดลองโดยใช้ Single Board เพียงอย่างเดียวได้ แต่วิธีการนี้เหมาะสำหรับการศึกษาโปรแกรมที่มีขนาดเล็กๆ เนื่องจากต้องใช้วิธีการป้อนรหัสคำสั่งของโปรแกรม ให้กับ Single Board โดยตรง ซึ่งถ้าโปรแกรมมีความยาวมากๆจะทำให้เสียเวลาในการป้อนโปรแกรมมาก และเมื่อมีการแก้ไขการทำงานของโปรแกรมโดยการเพิ่มเติมหรือลดคำสั่งแต่ละครั้งก็จำเป็นต้องป้อนรหัสคำสั่งโปรแกรมใหม่ด้วย

2. ใช้วิธีการพัฒนาโปรแกรมด้วย REMOTE-Z80 ซึ่งวิธีการนี้ ต้องทำการเชื่อมต่อ Single Board กับคอมพิวเตอร์ PC ทางพอร์ตอนุกรม RS232 ด้วย ซึ่งวิธีการนี้จะสะดวกมาก เนื่องจากสามารถเขียนSource Code โปรแกรมเก็บไว้ในคอมพิวเตอร์ PC และทำการบันทึกเป็นแฟ้มเก็บไว้ได้ และเมื่อต้องการแก้ไขหรือเปลี่ยนแปลงการทำงานของโปรแกรมก็สามารถแก้ไขหรือดัดแปลง Source Code แล้วทำการสั่งแปลโปรแกรมนั้นให้ได้เป็นรหัสคำสั่ง แบบ Intel HEX จากนั้นจึงสั่ง Download ข้อมูล Intel HEX นั้นลงมายังหน่วยความจำของ Single Board เพื่อสั่ง Run ได้

   โดยวิธีการนี้ต้องติดตั้งโปรแกรม Procomm Plus บนเครื่องคอมพิวเตอร์ PC เพื่อใช้ในการติดต่อสื่อสารระหว่าง Single Board กับ คอมพิวเตอร์ PC ด้วย นอกจากนี้แล้ว ยังต้องติดตั้งโปรแกรม Assembler ของ MCS51 เพื่อใช้สำหรับเขียนโปรแกรมและแปลโปรแกรมให้เป็นรหัสคำสั่งแบบ Intel HEX เพื่อใช้ Download ให้กับหน่วยความจำของ Single Board เพื่อสั่ง Run ในภายหลังด้วย

การเขียนโปรแกรมภาษา Assembly ตามข้อกำหนดของโปรแกรม “Cross32 V4.0”

โปรแกรม “Cross-32 Meta-Assembler V4.0” หรือเรียกสั้นๆว่าโปรแกรม “Cross32 V4.0” นั้นจัดเป็นโปรแกรมครอสแอสแซมเบอร์(Cross-Assembler) ถูกพัฒนาขึ้นโดย “Data Sync Engineering” ซึ่งเป็นบริษัทผู้ผลิตโปรแกรมสำหรับสนับสนุนการพัฒนางานทางด้านไมโครคอนโทรลเลอร์ขนาดเล็กอีกรายหนึ่งของอเมริกาซึ่งโปรแกรมตัวนี้มีจุดเด่นที่น่าสนใจตรงที่เป็นโปรแกรมจำพวก Assembler ประเภท Universal Cross-Assembler ซึ่งสามารถใช้แปลคำสั่งของ CPU ได้มากกว่า 50 ตระกูล รวมทั้ง CPU ตระกูล MCS51 และ Z80 ด้วย ซึ่งการที่โปรแกรมชุดนี้สามารถใช้งานกับ CPU ได้มากมายหลายตระกูลและหลายๆเบอร์นั้น จะเป็นผลดีอย่างมากกับผู้ศึกษาไมโครคอนโทรลเลอร์ ที่มีความสนใจและต้องการใช้งานไมโครคอนโทรลเลอร์หลายๆเบอร์ โดยไม่ต้องการมุ่งเน้นหรือหยุดอยู่ที่เบอร์ใดเบอร์หนึ่งโดยเฉพาะ ทั้งนี้ก็เพราะว่าจะได้มุ่งเน้นศึกษารูปแบบและข้อกำหนดต่างๆในการเขียนโปรแกรม Source Code ภายใต้ข้อกำหนดของโปรแกรม Cross32 V4.0 เพียงรูปแบบเดียว เมื่อต้องการจะเปลี่ยนไปใช้งานไมโครคอนโทรลเลอร์เบอร์อื่นๆอีกก็สามารถเขียนโปรแกรมได้ตามรูปแบบเดิมที่มีความคุ้นเคยอยู่ก่อนแล้ว เพียงแต่ต้องมีการปรับเปลี่ยนบ้างอีกเล็กน้อย คือ ส่วนของรหัสคำสั่งที่เป็นนีโมนิกส์ของ CPU แต่ละเบอร์เท่านั้นเอง ส่วนรูปแบบมาตรฐานอื่นๆ เช่น การกำหนดค่า Label การกำหนดค่าตัวเลข หรือ การใช้งานคำสั่งเทียมต่างๆ จะยังคงมีรูปแบบเหมือนกันไม่เปลี่ยนแปลง ซึ่งจัดเป็นข้อดี คือ จะได้ไม่ต้องไปคอยจดจำข้อกำหนดและรูปแบบของการเขียน Source Code สำหรับโปรแกรมแอสแซมเบอร์หลายๆชุดให้สับสนวุ่นวาย

สำหรับโปรแกรม Cross32 ชุดนี้ ได้รับการพัฒนาและปรับปรุงความสามารถอย่างต่อเนื่องเรื่อยมาจนถึงรุ่นปัจจุบันคือ รุ่น4 หรือ Version4.0 ซึ่งมีความอ่อนตัวในการใช้งานพอสมควร สามารถใช้งานได้กับระบบปฏิบัติการของ DOS และ Windows และยังสามารถใช้งานได้ทั้งการ Assembler แบบ Command Line เพียงอย่างเดียว คือ ใช้สำหรับทำหน้าที่สั่งแปลโปรแกรม Source Code คำสั่งภาษา Assembly ของไมโครคอนโทรลเลอร์ในรูปแบบของ Text File ที่เขียนไว้แล้ว หรือจะใช้งานในรูปแบบของโปรแกรม IDE คือ ใช้เขียน Source Code ภายใต้โปรแกรม Text Editor ของ Cross32 เอง พร้อมกับสั่งแปลโปรแกรมที่กำลังเขียนอยู่นั้นได้ทันที โดยไม่ว่าจะใช้งานในรูปแบบใดก็จะยังคงได้ Output File จากการแปลเป็นรหัสคำสั่ง (Opcode) ซึ่งมีลักษณะเป็นตัวเลขฐานสิบหก หรือ “Hex File” มาตรฐานตามข้อกำหนดของบริษัท Intel ซึ่งนิยมเรียกว่า “Intel Hex Format”(HEX) ซึ่งแบบฟอร์มในการเขียน Source Code ภาษา Assembly สำหรับใช้งานกับโปรแกรม Cross32 V4.0 นั้น รูปแบบโดยรวมแล้วจะเหมือนกับแบบฟอร์มในการเขียนโปรแกรมภาษา Assembly ตามหลักสากลที่กล่าวผ่านมาแล้วในข้างต้น แต่มีข้อกำหนดเพิ่มเติมบางประการในการเขียนโปรแกรมที่ควรทราบดังต่อไปนี้คือ

Line Format คือข้อกำหนดสำหรับการเขียนโปรแกรมในแต่ละบรรทัด ซึ่งควรจะเขียนโปรแกรมให้แต่ละบรรทัดมีคำสั่งเพียงคำสั่งเดียวเท่านั้น โดยถ้าต้องการเขียน Source Code ภายใต้โปรแกรม Text Editor ของ Cross32 เองนั้น ในแต่ละไฟล์สามารถเขียนโปรแกรมให้มีความยาวสูงสุดได้จำนวน 65,535 บรรทัด โดยในแต่ละบรรทัดจะเขียนข้อความได้มากสุด 255 ตัวอักษร และในแต่ละบรรทัดของโปรแกรมก็จะต้องแบ่ง Field ของโปรแกรมออกเป็น 4 ส่วน หลักๆ ดังต่อไปนี้

Lable (สัญลักษณ์)	Operation (คำสั่ง)	Operand (ตัวกระทำ)	Comment (คำอธิบาย)

ซึ่งในการเขียนโปรแกรมนั้น บางบรรทัดอาจมีครบทั้ง 4 Field บางบรรทัดอาจมีเพียง Field เดียว ก็ได้ ไม่แน่นอน ขึ้นอยู่กับลักษณะและรูปแบบของคำสั่งที่นำมาใช้เขียนในโปรแกรมเป็นหลัก

LABEL(ชื่อสัญลักษณ์) ในส่วนนี้จะมีหรือไม่มีก็ได้ แต่ถ้ามีจะต้องอยู่ใน Column แรกทางซ้าย และตัวอักขระตัวแรกของ LABEL นี้ควรต้องชิดขอบซ้ายของหน้ากระดาษเสมอ สำหรับชื่อของ LABEL นั้น สามารถตั้งได้ตามความต้องการ โดยใช้ตัวอักษรพิมพ์เล็กหรือใหญ่ก็ได้ ซึ่งจะมีความหมายเหมือนกัน ซึ่งอักขระที่จะสามารถนำมาใช้ตั้งเป็น ชื่อ LABEL นั้นตัวอักษรแรกต้องขึ้นต้นด้วย A-Z หรือ a-z หรือ "_" หรือ "." หรือ "?" เท่านั้น โดยตัวอักษรและสัญลักษณ์เหล่านี้สามารถนำมาใช้ตั้งเป็นชื่อได้ทั้งหมด โดยความยาวของชื่อ LABEL นั้นสามารถตั้งได้สูงสุดไม่เกิน 15 ตัวอักษร และ ต้องไม่ตั้งชื่อของ LABEL ให้ตรงกับชื่อคำสั่งหรือรีจิสเตอร์ของ CPU ด้วย และชื่อของ LABEL ต้องเรียงติดกันห้ามเว้นวรรค โดยชื่อ LABEL นั้นต้องปิดท้ายด้วยเครื่องหมายโคลอน (:) ที่ท้ายชื่อด้วยเสมอ แต่โปรแกรมจะไม่ถือเอา เครื่องหมายโคลอน(:) นี้เป็นส่วนของชื่อด้วย แต่ต้องใส่กำกับไว้ เพื่อบ่งบอกให้โปรแกรม Assembler ทราบว่า ชื่อนั้นคือ LABEL ตัวอย่างเช่น

```
HERE:   NOP
        DJNZ    B,HERE
หรือ
here:   NOP
        DJNZ    B,HERE
```

ซึ่งในการเขียนโปรแกรมภาษา Assembly ส่วนมากนั้นจะนิยมกำหนด ชื่อ LABEL ไว้เฉพาะในตำแหน่งของโปรแกรมที่ต้องการจะกระโดดไปทำงานหรืออ้างอิงถึงโดยคำสั่งอื่นๆหรืออาจเป็นตำแหน่ง เริ่มต้นของโปรแกรมย่อยต่างๆ ที่ต้องเรียกใช้ เพราะจะทำให้ง่ายต่อการจดจำและสื่อความหมายได้มากกว่าวิธีการที่ใช้การกำหนดค่าตำแหน่ง Address เป็นแบบตัวเลขโดยตรง และประการสำคัญเมื่อมีการแทรก หรือตัดทอนคำสั่งต่างๆในส่วนหนึ่งส่วนใดของโปรแกรมแล้วก็ยังสามารถที่จะนำโปรแกรมนั้นไปทำการแปลใหม่ได้อีกครั้งหนึ่ง ซึ่งโปรแกรม Assembler ก็จะสามารถทำการคำนวณหาค่าของตำแหน่ง Address ใหม่ที่ถูกต้องให้ได้ทันที

OPERATION(คำสั่ง) ในส่วนนี้จะต้องมีอยู่ในทุกบรรทัดที่เป็นโปรแกรมซึ่งใน Field นี้จะใช้เขียนคำสั่งภาษา Assembly ของ CPU (Mnemonic Code) เช่น MOV INC ADD SUBB เป็นต้น โดยคำสั่งที่เขียนขึ้นนั้นในส่วนที่เป็น Mnemonic เดียวกัน จะต้องเขียนให้ติดกันด้วยห้ามเว้นวรรค ถ้าหากมีการเว้นช่องว่างโปรแกรมจะถือว่าส่วนที่อยู่ด้านหลังช่องว่างของคำสั่ง (Mnemonic) นั้น เป็นค่าของตัวกระทำ (Operand) ซึ่งจะทำให้เกิดความผิดพลาดขึ้นได้ในการแปลโปรแกรม

OPERAND(ตัวกระทำ) ใน Field นี้อาจมีหรือไม่มีก็ได้ขึ้นอยู่กับรูปแบบของคำสั่ง (Mnemonic) ที่นำมาใช้เขียนโปรแกรมใน Field ของ Operation ซึ่งในส่วน Field ของ Operand นี้จะเป็นตัวบ่งบอก ให้ทราบถึงหน้าที่การกระทำของคำสั่ง(Mnemonic)ว่าจะให้ข้อมูลหรือตัวกระทำ(Operand) ใดมากระทำกัน ในคำสั่งบ้างซึ่งบางคำสั่ง เช่น RTS(Return From Subroutine) ก็จะไม่จำเป็นต้องมีค่าของ Operand เข้ามาเกี่ยวข้องในคำสั่ง แต่บางคำสั่งอาจมีตัวกระทำ Operand ในคำสั่ง 1 ตัว หรือมากกว่า ก็ได้ ถ้ามี Operand มากกว่า 1 ตัว จะแบ่งแยกแต่ละ Operand ด้วยเครื่องหมายคอมม่า (,) ตัวอย่างเช่น

ANL	A,#30H	ANL เป็นคำสั่ง(Mnemonic) ส่วน #30H เป็นตัวกระทำ
MOV	50H,A	คำสั่งนี้จะมีตัวกระทำ 2 ตัวคือ RAM ตำแหน่ง 50H และ A
RET		เป็นคำสั่งชนิดที่ไม่มีตัวกระทำ

COMMENT(คำอธิบาย) ในส่วนนี้จะมีหรือไม่มีก็ได้ไม่มีผลต่อการทำงานของโปรแกรมที่เขียนขึ้นแต่อย่างใดและโปรแกรม Assembler จะไม่นำส่วนคำอธิบาย หรือ Comment นี้ มาเกี่ยวข้องกับการแปลโปรแกรมด้วย ซึ่งส่วนคำอธิบายนี้จะนิยมเขียนกำกับไว้ในโปรแกรมเพื่อใช้เป็นคำอธิบายความหมายกันลืม เพื่อประโยชน์ในการกลับมาแก้ไขโปรแกรมเดิมที่เขียนไว้นานแล้วจะได้สามารถเข้าใจได้โดยง่าย โดยคำอธิบายนี้จะสามารถเขียนได้ยาวเท่าได้ก็ได้ไม่จำกัด เพียงแต่มีข้อแม้ว่า ถ้าเขียน Comment นี้อยู่ใน บรรทัดเดียวกับคำสั่งจะต้องเขียนใน Column สุดท้ายหลังจากจบรูปแบบของคำสั่งที่เขียนขึ้นแล้ว

โดยก่อนเริ่มต้นเขียนส่วน Comment นี้ต้องใส่เครื่องหมายเซมิโคลอน (;) นำหน้าไว้ด้วยเสมอ และห้ามไม่ให้ใส่ Comment ไว้หน้าคำสั่ง เพราะโปรแกรม Cross32 จะถือว่าข้อความที่อยู่ตามหลังเครื่องหมายเซมิโคลอน (;) เป็น Comment ทั้งหมดและจะไม่นำข้อความในบรรทัดนั้นมาแปลอีก

การกำหนดค่าตัวเลขในโปรแกรม (Integer Constant)

ในการกำหนดค่าตัวเลขให้กับโปรแกรมนั้น สามารถกำหนดค่าของตัวเลขที่จะใช้ในโปรแกรม ได้หลายรูปแบบ ไม่ว่าจะเป็นเลขฐานสอง ฐานแปด ฐานสิบ หรือ ฐานสิบหก โดยสรุปได้ดังนี้คือ

การใช้สัญลักษณ์นำหน้าตัวเลข การใช้สัญญลักษณ์นำหน้าระบบตัวเลขเพื่อใช้บ่งบอกถึงความแตกต่างของค่าตัวเลขที่ต้องการ ไม่ว่าจะเป็นเลขฐานแปด เลขฐานสิบ หรือ เลขฐานสิบหก ซึ่งมีข้อกำหนดดังนี้

- ถ้าค่าของตัวเลขขึ้นต้นด้วยเลขศูนย์จะหมายถึงเลขฐานแปด ซึ่งค่าตัวเลขที่ตามหลังจากศูนย์จะต้องมีค่าอยู่ระหว่าง 0-7 เท่านั้น
- ถ้าค่าของตัวเลขขึ้นต้นด้วยเลขศูนย์และตัวอักษร ‘x’ (0x) จะหมายถึงเลขฐานสิบหก ซึ่งสามารถตามด้วยตัวเลข 0-9 และตัวอักษร A-F เช่น 0x12,0xAB
- ถ้าค่าของตัวเลขขึ้นต้นด้วยเครื่องหมายดอลล่าร์ ($) จะหมายถึงเลขฐานสิบหก ซึ่งค่าที่ใช้เขียนตามหลังเครื่องหมาย $ สามารถเป็นได้ทั้งเลข 0-9 และตัวอักษร A-F ก็ได้ เช่น $1234 หรือ $FC00
- ถ้าค่าของตัวเลขนำหน้าด้วยค่าตัวเลขระหว่าง 1-9 จะหมายถึงเลขฐานสิบ เช่น 100,1234

ข้อควรระวัง ห้ามเขียนค่าตัวเลขฐานสิบโดยใส่เลขศูนย์นำหน้า เนื่องจากโปรแกรม Cross32 จะถือว่าเป็นค่าของเลขฐานแปด ซึ่งจะทำให้ได้ค่าที่ผิดไปจากความต้องการ เช่น 0255 จะไม่เท่ากับ 255

การใช้สัญลักษณ์ปิดท้ายตัวเลข การใช้สัญลักษณ์ตามหลังระบบตัวเลขเพื่อใช้บ่งบอกถึงความแตกต่างของค่าตัวเลขที่ต้องการก็เป็นอีกวิธีหนึ่งที่สามารถนำมาใช้ได้ ไม่ว่าจะเป็นเลขฐานสอง เลขฐานแปด เลขฐานสิบ หรือ เลขฐานสิบหก ซึ่งมีข้อกำหนดดังนี้

- ใช้ตัวอักษร 'B' ตามหลังค่าตัวเลขฐานสอง ซึ่งค่าของตัวเลขจะประกอบด้วยตัวเลขเพียง 2 จำนวน คือ 0 และ 1 เท่านั้น เช่น 10001011B
- ใช้ตัวอักษร 'O' หรือ 'Q' ตามหลังค่าตัวเลขฐานแปด ซึ่งค่าของตัวเลขที่ใช้นำหน้าตัวอักษร 'O' หรือ 'Q' นี้ต้องมีค่าไม่เกิน 8 คือมีค่าอยู่ระหว่าง 0-7 เท่านั้น
- ใช้ตัวอักษร 'D' ตามหลังค่าตัวเลขฐานสิบ ซึ่งค่าของตัวเลขที่ใช้นำหน้าตัวอักษร 'D' นี้ต้องมีค่าเป็นตัวเลขระหว่าง 0-9 เท่านั้น
- ใช้ตัวอักษร 'H' ตามหลังค่าตัวเลขฐานสิบหก ซึ่งค่าที่ใช้นำหน้าตัวอักษร 'H' นี้เป็นได้ทั้งค่าของตัวเลขระหว่าง 0-9 และตัวอักษร A-F และในกรณีที่ค่าของตัวเลขฐานสิบหกเริ่มต้นด้วยค่าที่ไม่ใช่ตัวเลข คือ A-F จะต้องใช้เลขศูนย์นำหน้าด้วย เช่น 1234H หรือ 0FFH เป็นต้น

การใช้ค่าสัญลักษณ์ของตัวอักษร ในการกำหนดค่าของสัญลักษณ์ตัวอักษรนั้นจะต้องกำหนดไว้ภายในเครื่องหมายฟันคู่ (") โดยค่าของตัวอักษรอาจเป็นตัวเดียวหรือหลายตัวก็ได้ เช่น "A","B","C","D" หรือ "Hello" ซึ่งเมื่อมีการกำหนดค่าในลักษณะแบบนี้ โปรแกรม Cross32 จะแปลค่าออกมาเป็นค่ารหัส ASCII ของตัวอักษรนั้นๆแทน เช่น "A" จะมีค่า 41H หรือ 65 นั่นเอง สำหรับเครื่องหมาย (") ไม่สามารถกำหนดค่าด้วยวิธีนี้ได้ แต่สามารถหลีกเลี่ยงไปใช้วิธีอื่นแทนได้โดยการกำหนดเป็นค่าของรหัส ASCII ของเครื่องหมายแทน คือ 22H

การกำหนดค่าทางคณิตศาสตร์และลอจิก ในการกำหนดค่าที่เป็นเครื่องหมายทางคณิตศาสตร์และการกระทำทางลอจิกนั้น จะแบ่งออกเป็น 2 ส่วนด้วยกัน คือ การกระทำทางคณิตศาสตร์จะเป็นตัวเลขขนาด 32 บิต แบบคิดเครื่องหมาย ส่วนการกระทำทางด้านลอจิกจะเป็นลักษณะจริงหรือเท็จ อย่างใดอย่างหนึ่งเท่านั้น และเพื่อให้ง่ายต่อการเข้าใจจะใช้ X และ Y เป็นตัวแปรสำหรับแสดงการทำงานของการกระทำทางคณิตศาสตร์และลอจิกดังต่อไปนี้

$	ใช้แทนค่าโปรแกรม Counter (PC) ณ ตำแหน่งนั้นๆ เช่น BRA $ หมายถึงให้กระโดดอยู่กับที่
{ }	หมายถึงให้กระทำในส่วนวงเล็บก่อน เช่น 4*("A"+26) จะนำค่า "A" บวกกับ 26 แล้วจึงนามาคูณกับ 4
!Y	ให้ตรวจสอบค่าตัวเลข Logic ทั้งหมดเป็น "0" ถ้าจริงค่า Y จะถูกเซ็ตเป็น "1"
~Y	กระทำ ONE'S Complement กับ Y

-Y	กระทำ TWO'S Complement กับ Y
+Y	จำนวนเต็มบวก
INV Y	กลับไบท์ ซ้าย,ขวา (Invert) เช่น 1234 = 3412
X*Y	X คูณ Y เช่น 0xfe * 16
X/Y	X หาร Y ไม่คิดเศษ เช่น 0xfe / 16
X÷Y	X หาร Y ใช้เฉพาะผลของเศษ เช่น 0xfe ÷ 16
X+Y	X บวก Y เช่น 40 + 20D
X-Y	X ลบ Y เช่น 40 - 20D
X<<Y	Shift ซ้าย X ด้วยจำนวนบิตใน Y โดย Y ต้องน้อยกว่า 32 เช่น 1234H << 8
X>>Y	Shift ขวา X ด้วยจำนวนบิตใน Y โดย Y ต้องน้อยกว่า 32 เช่น 1234H >> 8
X<Y	X น้อยกว่า Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 < 2
X<=Y	X น้อยกว่าหรือเท่ากับ Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 <= 2
X>Y	X มากกว่า Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 > 2
X>=Y	X มากกว่าหรือเท่ากับ Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 >= 2
X==Y	X มีค่าเท่ากับ Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 == 2
X!=Y	X ไม่เท่ากับ Y ถ้าเป็นจริงได้ผลลัพธ์เป็น “1” นอกนั้นเป็น “0” เช่น 0 != 2
X&Y	X กระทำ Logic AND แบบบิตต่อบิต กับ Y เช่น “3” & 15
X^Y	X กระทำ Logic XOR แบบบิตต่อบิต กับ Y เช่น 10B ^ 3
X | Y	X กระทำ Logic OR แบบบิตต่อบิต กับ Y เช่น 2 | 253
X&&Y	ค่าใน X กระทำ AND กับ ค่าใน Y ผลการ AND ถ้าค่าไม่เท่ากับ 0 ผลลัพธ์เป็น “1” เช่น 0 && 2
X||Y	ค่าใน X กระทำ OR กับ ค่าใน Y ผลการ OR ถ้าค่าไม่เท่ากับ 0 ผลลัพธ์เป็น “1” เช่น 0 || 2

คำสั่งเทียมในโปรแกรม Cross32 V4.0

ในการเขียนโปรแกรมภาษา Assembly นั้นจะมีความยุ่งยากกว่าโปรแกรมภาษาสูงอื่นๆพอสมควร โดยเฉพาะอย่างยิ่ง เมื่อต้องมีการใช้พื้นที่ของหน่วยความจำสำหรับเก็บข้อมูลชั่วคราวไว้ก่อน เมื่อต้องการ เรียกข้อมูลออกมาใช้จะเป็นการยุ่งยากอย่างมาก ว่าตำแหน่ง Address ใดถูกจัดสรรหน่วยความจำไว้สำหรับ เก็บค่าของอะไรบ้าง และยังอาจเกิดความสับสนเรียกใช้ผิดตำแหน่งได้ ทำให้เกิดความผิดพลาดขึ้นได้ง่าย ด้วยเหตุนี้โปรแกรมที่ทำหน้าที่เป็นตัว Assembler ต่างๆ จึงมีการคิดค้นสร้างคำสั่งอีกกลุ่มหนึ่งขึ้นมาใหม่ เพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรม โดยกลุ่มคำสั่งดังกล่าวนี้นิยมเรียกกันทั่วไปว่า "คำสั่งเทียม" ซึ่งคำสั่งเทียม นี้จะไม่ใช่คำสั่งสำหรับสั่งงาน CPU แต่จะถูกนำมาใช้เพื่อบ่งบอกให้โปรแกรม Assembler ได้ทราบถึงสิ่งที่จะกำหนดขึ้น เพื่อนำไปใช้ในการแปลโปรแกรมเท่านั้น ซึ่งคำสั่งเทียมของ Cross32 นั้นมีอยู่มากมาย แต่ในที่นี้จะขอกล่าวถึงเฉพาะในส่วนของคำสั่งเทียมที่มีความจำเป็นและใช้งานกันบ่อยให้ทราบพอสังเขปดังต่อไปนี้

ORG (Origin) เป็นคำสั่งเทียม ใช้ระบุค่าตำแหน่ง Address เพื่อบ่งบอกให้โปรแกรม Assembler ได้รับรู้ว่า สัญลักษณ์หรือคำสั่ง ที่เขียนต่อจากคำสั่ง ORG นั้นจะมีค่าตำแหน่ง Address เริ่มต้นตามค่า ที่กำหนดไว้จากคำสั่ง ORG นี้และค่า Address นั้นจะต่อเนื่องกันไปเรื่อยๆ เช่น

```
        ORG     0000H
MAIN:   MOV     SP,#60H; Define Stack Internal RAM
        .
        .
```

EQU (Equate) เป็นคำสั่งเทียมใช้สำหรับกำหนดค่าให้กับ ตัวแปรสัญลักษณ์(Symbol) ที่กำหนดไว้หน้าคำสั่ง EQU โดยโปรแกรม Assembler จะถือเอาค่า ของตัวแปรสัญลักษณ์ นั้นๆ ให้มีค่าตามที่ กำหนดไว้หลังคำสั่ง EQU ตลอดไปในการแปลคำสั่ง ดังนั้นทุกๆครั้งที่มีการอ้างถึง ตัวแปรสัญลักษณ์ ตัวนั้นๆเมื่อใด ก็จะมีความหมายเหมือนกับการอ้างถึงค่าที่กำหนดให้ทุกประการ ตัวอย่างเช่น

```
PORT_A:     EQU     20H
PORT_B:     EQU     PORT_A+1
PORT_C:     EQU     PORT_A+2
```

จากตัวอย่างการใช้คำสั่ง EQU ที่ผ่านมานั้น ในทุกๆตำแหน่งของโปรแกรมที่มีการอ้างถึง PORTA ก็จะมีความหมายเหมือนกับอ้างถึงค่า 20H เสมอ ส่วนค่าของ PORT_B และ PORT_C ก็จะมีค่าเป็น 21H และ 22H ตามลำดับ ถ้ามีการเปลี่ยนแปลงค่าของ PORT_A เป็นค่าอื่นก็จะส่งผลให้ PORT_B และ PORT_C ถูกเปลี่ยนแปลงค่าตามไปด้วย เราสามารถใช้คำสั่ง EQU กำหนดค่าให้กับ Symbol Name หนึ่งๆได้เพียงครั้งเดียว เท่านั้นในโปรแกรม ซึ่งลักษณะของคำสั่ง EQU จะเหมือนกับการกำหนดค่า Constant ในภาษาสูงนั่นเอง

DFB(Define Byte) เป็นคำสั่งเทียมใช้สำหรับกำหนดค่าของข้อมูลค่าคงที่ขนาด 8 บิต(Byte) ไว้ในโปรแกรม โดยคำสั่งนี้จะถูกนำมาใช้ประโยชน์ในการกำหนด ตาราง TABLE ต่างๆ ที่จะนำมาใช้ใน โปรแกรมโดยจำนวนข้อมูลแต่ละค่าจะถูกแบ่งแยกออกจากกันด้วยเครื่องหมายคอมม่า (,) ตัวอย่างเช่น

```
TAB_LED:    DFB     00000001B
            DFB     02H,04H,08H
TAB_MSG:    DFB     "ABC",00H
```

จากตัวอย่างที่ผ่านมาเมื่อสั่งให้โปรแกรม Assembler แปลโปรแกรมแล้ว ที่ตำแหน่ง Address ของ TAB_LED จะมีค่าคงที่อยู่ 4 ไบท์ คือ 01H,02H,04H และ 08H เรียงต่อเนื่องกันไป ส่วน TAB_MSG จะได้ค่าเป็นรหัส ASCII ของตัวอักษร ABC เรียงกันไป คือ 41H,42H,43H และปิดท้ายด้วย 00H

DWL(Define Word : LSB First) เป็นคำสั่งเทียมใช้สำหรับ สั่งกำหนดข้อมูลค่าคงที่ ขนาด 16 บิต(Word) โดยต้องกำหนด ค่าของ Byteต่ำก่อนแล้วตามด้วย Byte สูง และแบ่งแยกข้อมูลแต่ละ Word ด้วยเครื่องหมาย คอมม่า(,)

DFS (Define Storage) เป็นคำสั่งเทียมใช้สำหรับสงวน หรือ จองพื้นที่ของหน่วยความจำไว้ เท่ากับ จำนวนที่ระบุไว้หลังคำสั่ง DFS เช่น

```
                ORG     0000H
DSP_BUFF:       DFS     4
```

จะเป็นการบอกให้โปรแกรม Assembler ได้ทราบว่า Symbol ชื่อ DSP_BUFF นั้น ถูกกำหนดให้ มีขนาดของหน่วยความจำ 4ไบท์ โดยมีตำแหน่ง 0000H-0003H ซึ่งถ้ามีการอ้างถึง Symbol นี้ในโปรแกรม ก็จะมีความหมายเหมือนกับการอ้างถึงค่าตำแหน่ง Address 0000H-0003H ด้วย โดยถ้าใช้การอ้างแบบ 8 บิต เมื่อต้องการใช้ตำแหน่งที่ 2 ของ Symbol อาจใช้รูปแบบเป็น DSP_BUFF+1 แทนก็ได้ ซึ่งจะเป็นประโยชน์ มากในการเขียนโปรแกรมเพราะสามารถสื่อความหมายถึงหน้าที่ของหน่วยความจำได้ดีกว่าการอ้างเป็นค่า ตัวเลข และยังง่ายต่อการเปลี่ยนแปลงแก้ไขตำแหน่ง Address ในหน่วยความจำด้วย

END เป็นคำสั่งเทียมสำหรับบอกให้ Assembler ทราบว่าส่วนของโปรแกรมทั้งหมดสิ้นสุดลงก่อนหน้าตำแหน่งของคำสั่ง END นี้ ซึ่งต้องมีคำสั่งนี้ไว้ท้ายของโปรแกรมเสมอ และหากเขียนโปรแกรมต่อหลัง จากคำสั่ง END นี้แล้ว Assembler จะไม่แปลส่วนที่ต่อจาก END นี้ให้เลย

HOF (HEXADECIMAL Output Format) ใช้สำหรับกำหนดรูปแบบของ Output File โดยกำหนดได้ 3 ลักษณะ คือ Binary Output ,Intel Hex Output ,Motorola Hex Output ดังนี้

Binary File ประกอบด้วย BIN8 BIN16 BIN32 เป็น Output File แบบ Binary 8,16 และ 32 บิต

Intel Hex File ประกอบด้วย INT8 และ INT16 เป็น Output File แบบ Intel Hex ขนาด 8บิต และ 16 บิต ตามลำดับ ซึ่งเมื่อใช้กับ Z80 ต้องกำหนดเป็น INT8 เสมอ เนื่องจาก Z80 เป็น CPU ขนาด 8 บิต

Motorola Hex File ประกอบด้วย MOT8 ,MOT16 และ MOT32 เป็น Output File แบบ Motorola Hex ขนาด 8บิต 16 บิต และ 32 บิต ตามลำดับ

ตัวกระทำทางคณิตศาสตร์และลอจิก

สำหรับค่าตัวเลขต่างๆในการเขียนโปรแกรมเราสามารถกำหนดเป็น เลขฐานสอง(Binary) ฐานแปด (Octal) ฐานสิบ(Decimal) และ ฐานสิบหก(Hexadecimal) โดยกำหนดตัวอักษร B,O,D และ H ต่อท้ายตัวเลขตามลำดับ เช่น

```
01011B      ; Binary
072O        ; Octal
20D         ; Decimal
0A5H        ; Hex
```

สำหรับวิธีการเรียกใช้งาน โปรแกรม Complier Cross32 V4.0 เพื่อให้ทำหน้าที่แปลโปรแกรมภาษา Assembly ที่เขียนไว้แล้วให้เป็น File แบบ "Intel HEX Format" (HEX) มีรูปแบบดังนี้

Files ที่ใช้ Run ชื่อ C32D4CL.EXE
Files Table สำหรับ CPU MCS51 ชื่อ 8051.TBL
Files Table สำหรับ CPU Z80 ชื่อ Z80.TBL

ข้อกำหนดในการใช้งาน

เมื่อจะเริ่มเขียนโปรแกรม 2 บรรทัดแรกของโปรแกรมต้องกำหนดคำสั่งเทียบเบอร์ CPU และชนิดของ Output File ให้กับโปรแกรมเสียก่อนเนื่องจากโปรแกรม Cross32 V4.0 สามารถใช้งานกับ CPU ได้มากมายหลายเบอร์และสร้าง Output File ได้หลาย Format สำหรับในกรณีที่ใช้กับ CPU ตระกูล MCS51 หรือ Z80 ให้กำหนดดังนี้คือ

กรณี MCS51

```
CPU    “8051.TBL”        ; แปลเป็น Code คำสั่งของ MCS51
HOF    “INT8”            ; สร้าง Output File เป็นแบบ Intel Hex 8 Bit
```

กรณี Z80

```
CPU    “Z80.TBL”         ; แปลเป็น Code คำสั่งของ Z80
HOF    “INT8”            ; สร้าง Output File เป็นแบบ Intel Hex 8 Bit
```

สำหรับวิธีการสั่ง Complier สามารถทำได้โดยมีรูปแบบดังนี้คือ

C32D4CL FILENAME.ASM -Option FILENAME.OUT

เมื่อ

C32D4CL เป็น File Run ของ Cross32 V4.0

FILENAME.ASM เป็นชื่อ File ต้นฉบับ ภาษา Assembly ที่ต้องการนำมาแปล

Option เป็น ข้อกำหนดว่าต้องการ Output File เป็นแบบใด โดยกำหนดได้ 3 แบบคือ

-H หมายถึงให้ทำการแปลเป็น Output File แบบ HEX โดยรูปแบบของ Hex จะขึ้นอยู่กับข้อกำหนดในโปรแกรมที่เขียนขึ้นที่กำหนดไว้ใน HOF

-L หมายถึง ให้ทำการแปลเป็น Output File แบบ Listing เพื่อตรวจสอบ Code และตำแหน่ง Address ต่างๆของคำสั่งในโปรแกรม

-E หมายถึงให้สร้าง Output File เป็นแบบ Error Report โดยจะแสดงการ Error ต่างๆเป็น File ไว้00

FILENAME.OUT หมายถึง ชื่อ Output File ที่ต้องการให้สร้างขึ้น โดยสามารถกำหนดเป็นอะไรก็ได้ แต่อย่างไรก็ตามการตั้งชื่อควรกำหนดให้เป็นมาตรฐานโดยการตั้งชื่อให้เหมือนกับ FILENAME.ASM แล้วเปลี่ยนเฉพาะนามสกุลตามชนิดของ Output File เช่น

ถ้ากำหนด Option -H ควรกำหนดเป็น FILENAME.HEX

ถ้ากำหนด Option -L ควรกำหนดเป็น FILENAME.LST

ถ้ากำหนด Option -E ควรกำหนดเป็น FILENAME.ERR

ตัวอย่างเช่น

```
C32D4CL   TEST.ASM -L TEST.LST          ; สร้าง Listing File
C32D4CL   TEST.ASM -H TEST.HEX          ; สร้าง Hex File
C32D4CL   TEST.ASM -E TEST.LST          ; สร้าง Error File
```

ขอให้พึงระลึกไว้เสมอว่า คำสั่งเทียม นี้เป็นคำสั่งเพิ่มเติมที่โปรแกรม Assembler สร้างขึ้นมาเพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรม ซึ่งรูปแบบของคำสั่งเทียมต่างๆเหล่านี้ โปรแกรม Assembler แต่ละโปรแกรมอาจมีมากน้อยไม่เท่ากัน และรูปแบบในการใช้งานคำสั่งเทียมต่างๆ ก็จะไม่เหมือนกันด้วย ขึ้นอยู่กับข้อกำหนด ของโปรแกรม Assembler ที่จะนำมาใช้ในการแปลคำสั่งโปรแกรม ดังนั้นในการที่จะ เขียนโปรแกรมแต่ละครั้ง ควรศึกษาถึงรูปแบบและข้อกำหนดของคำสั่งเทียมต่างๆให้เข้าใจเสียก่อน เพื่อที่จะได้เขียนโปรแกรมให้ถูกต้องตามรูปแบบและข้อกำหนดที่โปรแกรม Assembler นั้นๆ กำหนดไว้ ตัวอย่างเช่น เมื่อเขียนโปรแกรมตามข้อกำหนดของโปรแกรม Cross32 V4.0 แล้วนำโปรแกรม นั้นไปให้โปรแกรม Assembler ตัวอื่นทำการแปลให้แล้ว อาจเกิดการ Error ขึ้นเป็นจำนวนมาก เนื่องจากรูปแบบและข้อกำหนดต่างๆของโปรแกรมทั้งสองตัวนี้จะไม่เหมือนกัน

ตัวอย่างโปรแกรมภาษาแอสแซมบลี้ของ MCS51

```
;/************************/;
;/*       EXAMPLE #1     */;
;/*  SHIFT LEFT LED FLAG */;
;/************************/;
;
P_DIGIT   EQU   0E000H            ; 8255 SYSTEM PORT A
P_SEGM    EQU   0E001H            ; 8255 SYSTEM PORT B
          ;
          CPU   "8051.TBL"        ; CPU MCS51
          HOF   "INT8"            ; Intel HEX 8 Bit
          ;
          ORG   8000H             ; Start Program 8000H
          ;
START:    MOV   DPTR,#P_DIGIT     ; Port-PA
          MOV   A,#6
          MOVX  @DPTR,A           ; LED FLAG ACTIVE
          INC   DPTR              ; PORT SEGMENT
          MOV   A,#1              ; DATA LED FIRST

LOOP:     MOVX  @DPTR,A           ; OUT SEGMENT

          MOV   R2,#80H           ; DELAY
DEL1:     MOV   R3,#0
          DJNZ  R3,$
          DJNZ  R2,DEL1
          RL    A                 ; ROTATE DATA TO LEFT
          SJMP  LOOP

          END
```

ตัวอย่างโปรแกรมภาษาแอสแซมบลี้ของ Z80

```
;/************************/;
;/*      EXAMPLE #1      */;
;/* ROTATE LEFT LED FLAG */;
;/************************/;
;
P_DIGIT   EQU   00H                ; 8255 SYSTEM PORT A
P_SEGM    EQU   01H                ; 8255 SYSTEM PORT B

          CPU   "Z80.TBL"
          HOF   "INT8"
          ;
          ORG   8000H              ; Start Program 8000H
          ;
START:    LD    A,6                ; DIGIT 6 = LOW
          OUT   (P_DIGIT),A
          LD    D,1                ; First data
          ;
LOOP:     LD    A,D                ; Send Data to Segment
          OUT   (P_SEGM),A
          ;
          LD    BC,8000H           ; Delay
DEL:      DEC   BC
          LD    A,B
          OR    C
          JR    NZ,DEL
          ;
          RLC   D                  ; Data move left
          JR    LOOP

          END
```