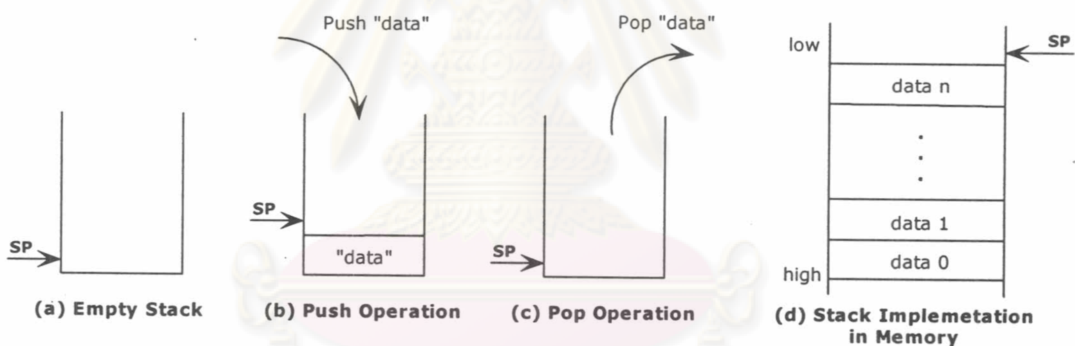
บทที่ 3
รายละเอียดของอุปกรณ์พื้นฐาน

บทนี้กล่าวถึงความรู้พื้นฐานเกี่ยวกับการทำงานแบบแสตกซึ่งเป็นกลไกการทำงานพื้นฐานของชุดคำสั่งแบบแสตก ต่อมากล่าวถึงรายละเอียดของชุดคำสั่งแบบแสตก SM1 และกล่าวถึงสถาปัตยกรรมของหน่วยประมวลผล C1 ที่เลือกใช้ในงานวิจัยนี้ในหัวข้อสุดท้าย

3.1 แสตกและการทำงานของหน่วยประมวลผล

แสตก (Stack) คือโครงสร้างข้อมูลแบบหนึ่งที่มีลักษณะการจัดเก็บข้อมูลที่อนุญาตให้ข้อมูลที่เข้ามาทีหลังออกก่อน (Last in First out, LIFO) ดังรูปที่ 3.1(a) โครงสร้างของแสตกมีทางเข้าออกทางเดียว ทำให้ข้อมูลที่พึ่งเข้ามาในแสตกต้องออกไปก่อน ดังนั้นการทำงานกับแสตกจึงทำกับข้อมูลบนสุดของแสตกผ่านทางตัวดำเนินการพื้นฐานสองตัวได้แก่ตัวดำเนินการดัน (Push operation) และดึง (Pop operation) ซึ่งการทำงานของแสตกและตัวดำเนินการทั้งสองจะแสดงไว้ในรูปที่ 3.1(b) และ (c)



รูปที่ 3.1 แสตกและตัวดำเนินการของแสตก

การนำเอาแสตกมาใช้ในระบบคอมพิวเตอร์นั้นจะใช้หน่วยความจำ (Memory) เป็นพื้นที่เก็บข้อมูลของแสตก การเข้าถึงข้อมูลบนแสตกกระทำผ่านการอ้างอิงด้วยตัวชี้แสตก (Stack pointer, SP) การดันข้อมูลลงแสตกคือการเขียนข้อมูลลงหน่วยความจำในตำแหน่งเลขที่อยู่ที่ SP ชี้อยู่ ส่วนการดึงข้อมูลเป็นการอ่านข้อมูลจากหน่วยความจำในเลขที่อยู่ที่ SP ชี้อยู่

โครงสร้างข้อมูลแบบแสตกมีบทบาทที่สำคัญมากกับการทำงานของคอมพิวเตอร์สองกลไกหลักได้แก่ กลไกในการหาค่านิพจน์ (Expression evaluation) และการจัดการโปรแกรมย่อย (Subroutine management)

3.1.1 กลไกในการหาค่านิพจน์

การหาค่านิพจน์ (Expression evaluation) ในคอมพิวเตอร์นั้น นิพจน์ (Expression) ในโปรแกรมจะถูกแปลงจากรูปแบบสัญกรณ์เติมกลาง (Infix notation) เป็นรูปแบบสัญกรณ์เติมหลัง (Postfix notation) ซึ่งจะทำให้ง่ายต่อการคำนวณผลลัพธ์ ดังตัวอย่าง

X = A + B * C

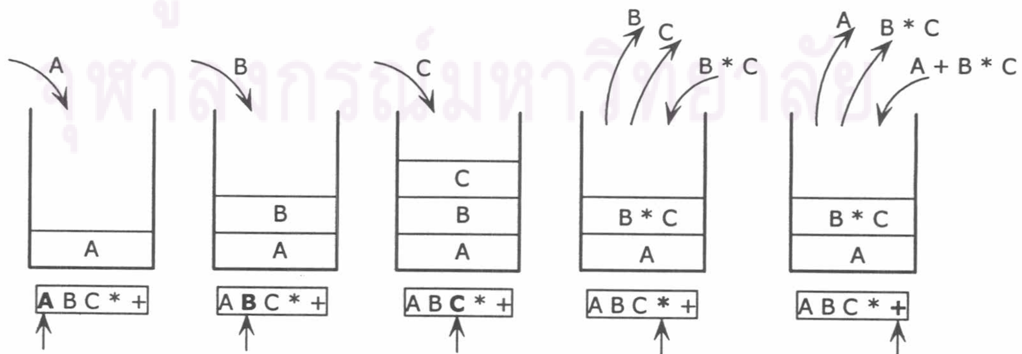
ตัวแปร B กับ C ต้องคูณกันก่อนแล้วจึงนำผลลัพธ์ไปบวกเข้ากับ A ซึ่งถ้าใช้การคิดแบบสัญกรณ์เติมกลางจะทำให้การคำนวณเป็นไปได้ยาก เนื่องจากถ้าพิจารณาทีละตัวจากซ้ายไปขวาทำให้การหาคำตอบของนิพจน์เป็น

X = (A + B) * C

ซึ่งทำให้ผลการคำนวณได้ผลลัพธ์ที่ผิด แต่ถ้าเปลี่ยนการคำนวณมาใช้แบบสัญกรณ์เติมหลังจะทำให้การคำนวณเป็นไปได้อย่างถูกต้องและง่ายขึ้นดังนี้

X = A B C * +

ซึ่งโครงสร้างข้อมูลแบบแสตกจะรองรับการคำนวณแบบสัญกรณ์เติมหลังได้เป็นอย่างดี แสดงได้ดังรูปที่ 3.2 การทำงานจะพิจารณาทีละตัวในนิพจน์ เมื่อเจอตัว A ระบบจะดันลงแสตก เช่นเดียวกับเมื่อเจอตัว B และ C ระบบจะดันค่าลงแสตก แต่เมื่อตัวที่พิจารณาอยู่เป็นตัวดำเนินการคูณ ระบบจะดึงค่าสองค่าที่อยู่บนแสตก (B และ C) ออกมาและคูณค่าทั้งสองเข้าด้วยกัน หลังจากนั้นจะดันผลลัพธ์กลับลงแสตก หลังจากนั้นเมื่อเจอตัวดำเนินการบวกจะทำเช่นเดียวกับตัวดำเนินการคูณ ก็จะได้ผลลัพธ์ของนิพจน์นี้ออกมา



รูปที่ 3.2 การใช้แสตกในการคำนวณแบบสัญกรณ์เติมหลัง

3.1.2 กลไกในการเรียกโปรแกรมย่อย

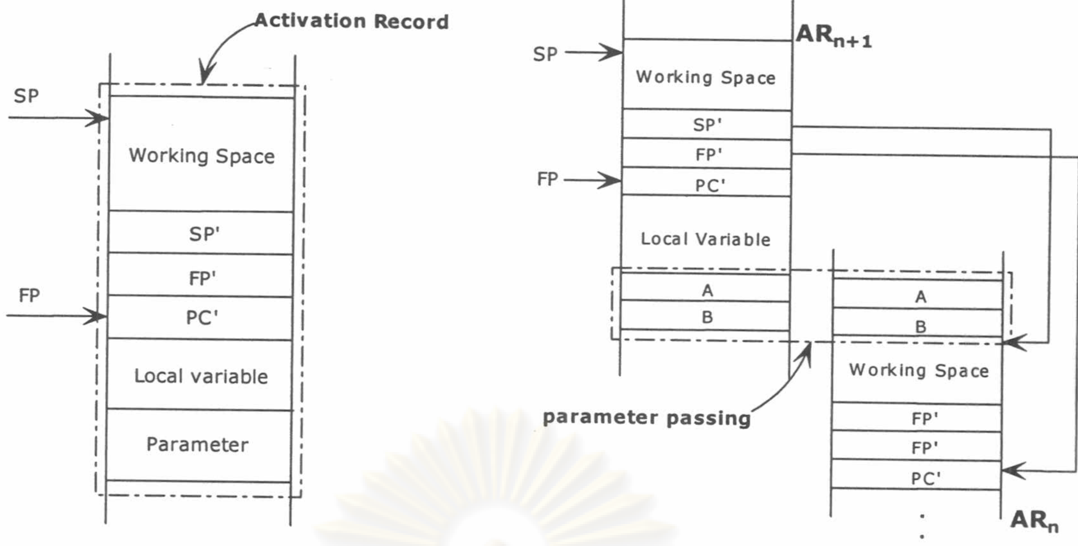
การเรียกโปรแกรมย่อยเป็นกลไกที่สำคัญในการพัฒนาโปรแกรม ทุกครั้งที่มีการเรียกโปรแกรมย่อยจำเป็นต้องมีกลไกในการจัดการสิ่งต่างๆ ดังต่อไปนี้

1. กลไกในการจัดการให้สามารถกลับไปทำงานต่อที่โปรแกรมหลักหลังจากเสร็จสิ้นการทำงานในโปรแกรมย่อย
2. การส่งพารามิเตอร์ (Parameter) จากโปรแกรมหลักไปยังโปรแกรมย่อย
3. จองพื้นที่และเข้าถึงตัวแปรเฉพาะที่ (Local variable) ที่อยู่ภายในโปรแกรมย่อย
4. จัดสรรพื้นที่แสตกที่ใช้ในโปรแกรมย่อยนั้นๆ ไม่ให้ไปซ้อนทับกับพื้นที่แสตกของโปรแกรมย่อยอื่น และจัดการคืนพื้นที่แสตกนั้นๆ หลังจากที่โปรแกรมย่อยทำงานเสร็จสิ้น

กลไกที่ใช้ในการจัดการสิ่งเหล่านี้เรียกว่า “แอคทิเวชันเรคคอร์ด” (Activation record) ซึ่งเป็นโครงสร้างข้อมูลที่ทำงานบนแสตก โครงสร้างของแอคทิเวชันเรคคอร์ดแบ่งออกเป็น 4 ส่วนสำหรับเก็บค่าต่างๆ ที่ใช้ในการกลไกการเรียกโปรแกรมย่อย ดังรูปที่ 3.3(a) ได้แก่ (1) เก็บสถานะการคำนวณ (Computation state) ซึ่งจะเก็บค่า PC, SP และ FP สำหรับใช้ในการกลับไปทำงานที่โปรแกรมหลัก (2) ค่าพารามิเตอร์ที่ส่งมาจากโปรแกรมหลัก (3) ตัวแปรเฉพาะที่ และ (4) พื้นที่แสตกที่ใช้ในการทำงานภายในโปรแกรมย่อย

การทำงานของแอคทิเวชันเรคอร์ดอาศัยตัวชี้สองตัว ได้แก่ ตัวชี้กรอบ (Frame pointer, FP) และตัวชี้แสตก (Stack pointer, SP) โดยที่ตัวชี้กรอบใช้ในการอ้างถึงแอคทิเวชันเรคคอร์ดนั้นๆ ส่วนตัวชี้แสตกใช้ในการเข้าถึงข้อมูลในแสตกที่อยู่ในแอคทิเวชันเรคคอร์ดนั้นๆ

กลไกการเรียกโปรแกรมย่อยกำหนดให้แอคทิเวชันเรคคอร์ดหนึ่งอันแทนการเรียกโปรแกรมย่อยหนึ่งครั้ง เมื่อมีการเรียกโปรแกรมย่อยทุกครั้งจะเกิดการสร้างแอคทิเวชันเรคคอร์ดขึ้นมาใหม่ดังในรูปที่ 3.3(b) ถ้ามีการเรียกโปรแกรมย่อยเกิดขึ้นขณะที่อยู่ที่ AR_n ระบบจะสร้าง AR_{n+1} ขึ้นใหม่ และเก็บสถานะการคำนวณซึ่งประกอบด้วยค่าเลขที่อยู่กลับ (Return Address) ของโปรแกรมย่อย เก็บค่าตัวชี้กรอบ (FP) และตัวชี้แสตก (SP) ของแอคทิเวชันเรคคอร์ดของโปรแกรมหลัก เมื่อโปรแกรมย่อยทำงานเสร็จ จะกลับมาทำงานในโปรแกรมหลัก ณ ตำแหน่งที่เก็บในค่าเลขที่อยู่กลับ และตำแหน่งของแอคทิเวชันเรคอร์ดของโปรแกรมหลัก (AR_n)



รูปที่ 3.3 โครงสร้างของแอคทิเวชันเรคคอร์ดและกลไกการเรียกโปรแกรมย่อย

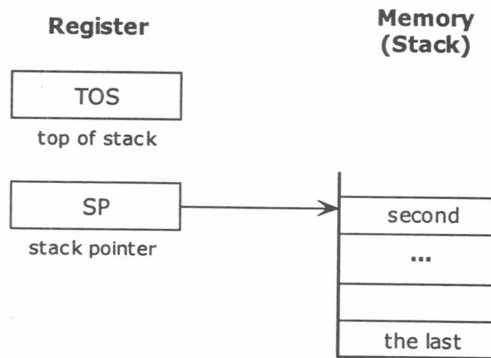
การส่งพารามิเตอร์เข้าไปในโปรแกรมย่อยใช้การซ้อนทับกันของแอคทิเวชันเรคอร์ดทั้งสองอัน พารามิเตอร์ที่ถูกส่งมาจากแอคทิเวชันเรคอร์ดเดิมกลายมาเป็นตัวแปรเฉพาะที่ของแอคทิเวชันเรคอร์ดอันถัดไป ส่วนการอ้างถึงตัวแปรเฉพาะที่นั้น จะอ้างอิงกับตัวชี้เฟรม ทำให้สามารถจัดการกับตัวแปรเฉพาะที่ของโปรแกรมย่อยได้ง่ายและเป็นระบบ

แอคทิเวชันเรคอร์ดทำให้การจัดการกับโปรแกรมย่อยสะดวกขึ้น อีกทั้งยังสนับสนุนการเขียนโปรแกรมแบบเรียกซ้ำ (Recursive call) อีกด้วย

3.1.3 แสตกแคชชิ่ง

แสตกแคชชิ่ง (Stack caching) [22] เป็นหลักการหนึ่งที่ใช้ในการเพิ่มความเร็วการเข้าถึงข้อมูลในแสตก โดยมีแนวคิดมาจากการสังเกตพฤติกรรมการเข้าถึงข้อมูลในแสตก ข้อมูลที่จะถูกใช้ก่อนคือข้อมูลที่พึ่งเข้ามาเก็บในแสตก ดังนั้นการเก็บข้อมูลบนแสตกไว้ในเรจิสเตอร์ที่สามารถเข้าถึงได้เร็วกว่าหน่วยความจำแล้วจะลดเวลาที่ใช้ในการเข้าถึงแสตกได้ เนื่องจากไม่จำเป็นต้องเข้าถึงหน่วยความจำทุกครั้ง

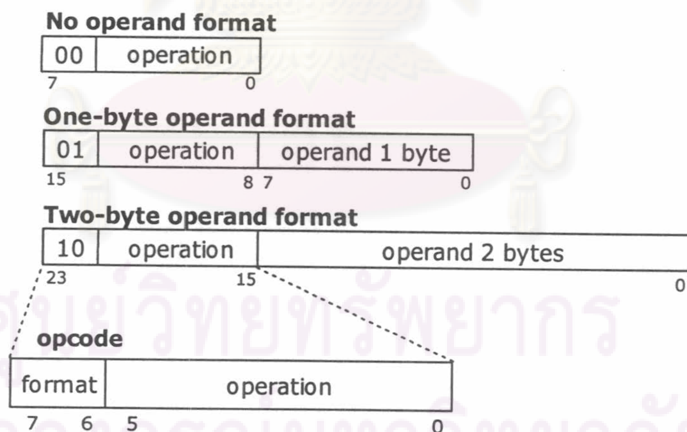
รูปที่ 3.4 แสดงหลักการของแสตกแคชชิ่งคือการนำเอาข้อมูลตัวบนสุดของแสตกมาเก็บไว้ในเรจิสเตอร์แทนการเก็บลงในหน่วยความจำที่เป็นแสตก และข้อมูลที่ถูกเรจิสเตอร์ SP ชี้อยู่เป็นข้อมูลตัวที่สองบนแสตก ซึ่งเป็นการลดการเข้าถึงข้อมูลแรกสุดในแสตกทำให้การทำงานเร็วขึ้น



รูปที่ 3.4 หลักการของแสตกแคชชิ่ง

3.2 ชุดคำสั่งแบบแสตก SM1

ชุดคำสั่ง SM1 เป็นชุดคำสั่งแบบแสตก (Stack-based instruction set) ประกอบไปด้วยคำสั่งแบบแสตกทั้งสิ้น 25 คำสั่ง รูปแบบของชุดคำสั่ง SM1 มีด้วยกัน 3 รูปแบบดังรูปที่ 3.5 โดยแบ่งตามขนาดของตัวถูกดำเนินการ (Operand) ได้แก่คำสั่งที่ไม่มีตัวถูกดำเนินการ (No operand format) คำสั่งที่มีตัวถูกดำเนินการขนาดหนึ่งไบต์ (One-byte operand format) และคำสั่งที่มีตัวถูกดำเนินการขนาดสองไบต์ (Two-byte operand format) ไบต์แรกของทุกๆ รูปแบบของชุดคำสั่งจะเป็นรหัสดำเนินการ (Operation code: opcode) และ 2 บิตแรกของรหัสดำเนินการระบุถึงรูปแบบของคำสั่ง (Opcode type) นั้นๆ ดังตารางที่ 3.1



รูปที่ 3.5 รูปแบบของชุดคำสั่งแบบแสตก SM1

ตารางที่ 3.1 รายละเอียดของรูปแบบคำสั่งของ 2 บิตแรกในรหัสดำเนินการ

Opcode type	Format	Bytecode size (Byte)
00	No operand format	1
01	One-byte operand format	2
10	Two-byte operand format	3
11	Not defined	-

คำสั่งทั้ง 25 คำสั่งของชุดคำสั่ง SM1 นั้นแบ่งออกเป็น 4 ประเภทด้วยกันได้แก่กลุ่มคำสั่งการคำนวณและตรรกะ (Arithmetic and logic instructions) กลุ่มคำสั่งย้ายข้อมูล (Data transfer instruction) กลุ่มคำสั่งควบคุม (Control flow instruction) และกลุ่มคำสั่งจัดการตัวแปรเฉพาะที่ (Local variable management)

รายละเอียดของการทำงานของคำสั่งแสดงไว้ในตารางที่ 3.4 การทำงานของคำสั่งจะใช้เรจิสเตอร์ที่จำเป็นทั้งสิ้น 5 ตัว ซึ่งตารางที่ 3.2 แสดงสัญลักษณ์และหน้าที่การทำงานของเรจิสเตอร์ต่างๆ เหล่านั้นไว้ การทำงานของคำสั่งใช้แนวคิดของแสตกแคชชิ่งในการเก็บข้อมูลบนสุดไว้ในเรจิสเตอร์ TOS เพื่อเพิ่มความเร็วในการทำงานของคำสั่ง

ตารางที่ 3.2 สัญลักษณ์และหน้าที่ของเรจิสเตอร์ที่ใช้ในชุดคำสั่ง SM1

สัญลักษณ์	**หน้าที่การทำงานของเรจิสเตอร์**
PC	ใช้ในการอ้างถึงคำสั่งในหน่วยความจำ (Program counter)
TOS	ใช้ในการเก็บข้อมูลบนสุดในแสตก (Top of stack register)
TMP	ใช้ในการคำนวณทั่วไป
SP	ตัวชี้แสตก (Stack pointer)
FP	ตัวชี้กรอบ (Frame pointer)

ในตารางที่ 3.4 การดำเนินการพื้นฐานของคำสั่งรหัสไบต์คือการดัน (Push) และดึง (Pop) ข้อมูลจากแสตกขึ้นมาประมวลผล โดยการทำงานของการดันและดึงแสดงได้ดัง

ตารางที่ 3.3 การดำเนินการดันและดึงของคำสั่งรหัสไบต์

Operator	RTL Operation
PUSH	MEM[SP] ← TOS; SP ← SP-1; TOS ← Data-to-push;
POP	Operate TOS as Pop-data; TOS ← MEM[SP+1]; SP ← SP + 1;

เมื่อคำสั่งดังกล่าวต้องการดันข้อมูลลงแสตก ต้องนำค่าที่อยู่ใน TOS ไปเก็บในแสตกส่วนที่เป็นหน่วยความจำก่อนแล้วจึงนำข้อมูลที่ต้องการดันถูกเก็บลงไปที่ TOS (จากหลักการแสตกแคชชิ่ง) และลดค่า SP ลงไปหนึ่งเพื่อชี้ไปยังแสตกช่องที่ว่างถัดไป

สำหรับการดึงข้อมูลนั้น ข้อมูลบนสุดของแสตกถูกเก็บไว้ใน TOS จึงสามารถดำเนินการกับข้อมูลดังกล่าวได้ทันที และหลังจากเสร็จสิ้นการดำเนินการแล้วต้องไปนำค่าที่อยู่ในแสตกอันดับถัดไปมาเก็บใน TOS แทนข้อมูลที่ถูกใช้ไป

ตารางที่ 3.4 รายละเอียดของชุดคำสั่งแบบแสตก SM1

Mnemonic	Operand	Description	Operation	Format
Arithmetic and logic instructions				
INC	-	Increment the top of stack	TOS ← TOS+1;	No operand
DEC	-	Decrement the top of stack	TOS ← TOS-1;	No operand
SHL	-	Shift the top of stack left one bit	TOS ← TOS<<1;	No operand
SHR	-	Shift the top of stack right one bit	TOS ← TOS>>1;	No operand
ADD	-	Add two data in the top of stack	TOS ← MEM[SP+1]+TOS; SP ← SP+1;	No operand
SUB	-	Sub two data in the top of stack	TOS ← MEM[SP+1]-TOS; SP ← SP+1;	No operand
AND	-	And two data in the top of stack	TOS ← MEM[SP+1]&TOS; SP ← SP+1;	No operand
OR	-	Or two data in the top of stack	TOS ← MEM[SP+1]| TOS; SP ← SP+1;	No operand
XOR	-	Xor two data in the top of stack	TOS ← MEM[SP+1]^TOS; SP ← SP+1;	No operand
LT	-	Less than comparison	if (MEM[SP+1] < TOS) TOS ← 1; SP ← SP+1;	No operand
EQ	-	Equal comparison	if (MEM[SP+1] == TOS) TOS ← 1; SP ← SP+1;	No operand
Data transfer instructions				
LD	-	Load data from the memory	TOS ← MEM[TOS];	No operand
ST	-	Store top of stack to the memory	TMP ← MEM[SP+1]; MEM[TMP] ← TOS; TOS ← MEM[SP+2]; SP ← SP+2;	No operand
LIT16	lit16bit	Push the literal 16 bit to the top of stack	MEM[SP] ← TOS; TOS ← lit16bit; SP ← SP-1;	Two-byte
LIT8	lit8bit	Push the literal 8 bit to the top of stack	MEM[SP] ← TOS; TOS ← lit8bit; SP ← SP-1;	One-byte
Control flow instructions				
JMP	OFFSET	Unconditional jump	PC ← PC + offset;	Two-byte
JT	OFFSET	Jump if true (TOS = 1)	if(TOS == 1) PC ← PC+OFFSET; else PC ← PC+1; TOS ← MEM[SP+1]; SP ← SP+1;	Two-byte
JF	OFFSET	Jump if fault (TOS != 1)	if(TOS == 0) PC ← PC+OFFSET else PC ← PC+1; TOS ← MEM[SP+1]; SP ← SP+1;	Two-byte
CALL	ADS	Call subroutine	MEM[SP] ← PC; PC ← ADS;	Two-byte

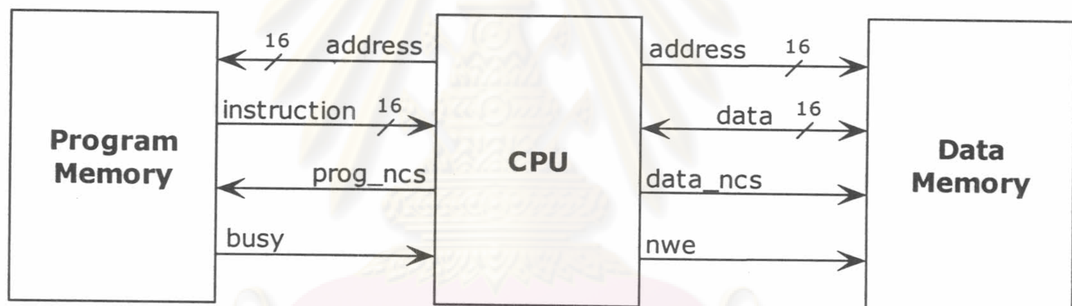
ตารางที่ 3.4 รายละเอียดของชุดคำสั่งแบบแสตก SM1(ต่อ)

Mnemonic	Operand	Description	Operation	Format
NEWFP	NLOCAL	Set the frame pointer (use after CALL)	MEM[SP+NLOCAL] ← MEM[SP]; MEM[SP+NLOCAL-1] ← FP; FP ← SP+NLOCAL; MEM[SP] ← TOS;	One-byte
NEWSP	NPARAM	Set the tack pointer (use after NEWFP)	TOS ← SP+NPARAM; SP ← FP-2;	One-byte
RET	-	Return from subroutine	TMP ← FP; SP ← TOS; FP ← MEM[TMP-1]; PC ← MEM[TMP]; TOS ← MEM[SP];	No operand
RETV	-	Return from subroutine with return value	TMP ← FP; SP ← TOS; FP ← MEM[TMP-1]; PC ← MEM[TMP]; TOS ← MEM[SP];	No operand
Local variable management				
GET	LOCAL	Get the local variable to the top of stack	MEM[SP] ← TOS; TOS ← MEM[FP+LOCAL]; SP ← SP+1;	One-byte
PUT	LOCAL	Set the local variable with top of stack	MEM[FP+LOCAL] ← TOS; TOS ← MEM[SP+1]; SP ← SP+1;	One-byte

3.3 รายละเอียดของหน่วยประมวลผล C1

หน่วยประมวลผล C1 ถูกพัฒนาขึ้นเพื่อใช้ในเว็บเซิร์ฟเวอร์แบบฝังตัวในงานวิจัยของเกริก [23] ซึ่งเดิมใช้ไมโครคอนโทรลเลอร์ MCS-51 ในการควบคุมระบบทั้งหมด และหน่วยประมวลผลดังกล่าวได้รับการทวนสอบโดยใช้หลักการของการทวนสอบเชิงรูปนัย (Formal verification) ในงานวิจัยของประพนธ์ [24] ในวิทยานิพนธ์นี้ได้ปรับปรุงการทำงานของคำสั่งบางประการให้มีประสิทธิภาพมากยิ่งขึ้น

หน่วยประมวลผล C1 นี้ (ต่อไปจะขอเรียกสั้นๆ ว่าหน่วยประมวลผล) เป็นหน่วยประมวลผลขนาด 16 บิต มีสถาปัตยกรรมแบบฮาร์วาร์ด (Harvard architecture) กล่าวคือมีบัสคำสั่ง (Instruction bus) และบัสข้อมูล (Data bus) ขนาด 16 บิตแยกจากกัน นอกจากนั้นขนาดของบัสเลขที่อยู่ (Address bus) ของคำสั่งและข้อมูลยังมีขนาด 16 บิต ทำให้ C1 สามารถเข้าถึงหน่วยความจำคำสั่ง และหน่วยความจำข้อมูลได้ 2^{16} คำ (Word) หน่วยประมวลผลดังกล่าวไม่มีการทำงานแบบสายท่อ (Pipeline) และไม่รองรับสัญญาณการขัดจังหวะ (Interrupt)



รูปที่ 3.6 สายสัญญาณระหว่างหน่วยประมวลผลและหน่วยความจำ

ภายในมีเรจิสเตอร์ขนาด 16 บิตสำหรับใช้ในการคำนวณทั่วไป (General register) จำนวน 4 ตัว ได้แก่เรจิสเตอร์ R0 – R3 นอกจากนั้นยังมีเรจิสเตอร์เซกเมนต์ (Segment register) ขนาด 8 บิต 1 ตัวสำหรับการอ้างอิงถึงหน่วยความจำในเซกเมนต์ที่ต้องการ และมีเรจิสเตอร์บ่งชี้ (Flag register) 3 ตัว ได้แก่ เรจิสเตอร์ตัวทด (Carry flag, c) เรจิสเตอร์บ่งชี้ศูนย์ (Zero flag, z) และเรจิสเตอร์เครื่องหมาย (Sign flag, s)

รายละเอียดสถาปัตยกรรมหน่วยประมวลผลแบ่งออกเป็นรายละเอียดของชุดคำสั่งและสถาปัตยกรรมของหน่วยประมวลผล

3.3.1 รายละเอียดชุดคำสั่ง

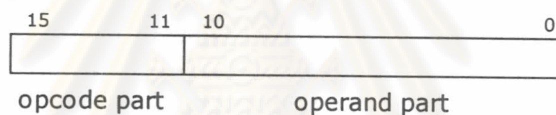
ชุดคำสั่งของหน่วยประมวลผลได้รับการปรับปรุงจากหน่วยประมวลผลที่ได้รับการทวนสอบในงานวิจัยของประพนธ์ [24] โดยการเพิ่มคำสั่งและแอดเดรสซิ่งโหมดเพื่อเพิ่มประสิทธิภาพ

ในการทำงาน คำสั่งมีด้วยกันทั้งสิ้น 32 คำสั่งแบ่งออกเป็น 3 ประเภทตามชนิดของการดำเนินการ ได้แก่ กลุ่มคำสั่งการคำนวณและตรรกะ (Arithmetic and logic instructions) กลุ่มคำสั่งการย้ายข้อมูล (Data transfer instructions) และกลุ่มคำสั่งการควบคุม (Control flow instructions)

ตารางที่ 3.5 สัญลักษณ์ที่ใช้ในการอธิบายถึงการทำงานของคำสั่ง

Symbol	Description
RS0, RS1	Register field refer to register file {R0, R1, R2, R3}
DISP	Displacement for index addressing mode
OFFSET	Offset address
ADS	Absolute address
IMM	Immediate value
PC	Program counter
C, Z, S	carry flag, zero flag and sign flag
MEM[ADDR]	Data in the memory in address ADDR

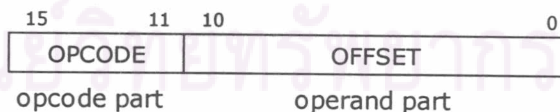
คำสั่งแต่ละคำสั่งมีขนาด 16 บิตเท่ากันทุกคำสั่ง โครงสร้างของคำสั่งประกอบด้วยส่วนประกอบหลัก 2 ส่วนได้แก่ส่วนรหัสดำเนินการ (Opcode) และส่วนตัวถูกดำเนินการ (Operand) โดยที่ส่วนรหัสดำเนินการมีขนาด 5 บิต และส่วนตัวถูกดำเนินการมีขนาด 11 บิต ดังแสดงในรูปที่ 3.7



รูปที่ 3.7 โครงสร้างของคำสั่ง

โครงสร้างของคำสั่งจะแบ่งตามรูปแบบของส่วนตัวถูกดำเนินการ ซึ่งถูกกำหนดโดยแอดเดรสซิ่งโหมด (Addressing mode) มีทั้งหมด 7 โหมด ได้แก่

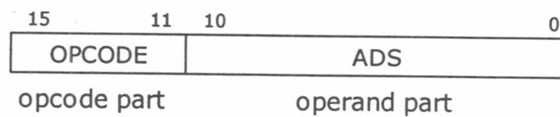
1. แบบสัมพัทธ์ (PC relative addressing mode)



รูปที่ 3.8 โครงสร้างของคำสั่งในโหมดสัมพัทธ์

เป็นแอดเดรสซิ่งโหมดสำหรับคำสั่งในกลุ่มคำสั่งการควบคุม ใช้ในการกระโดดแบบสัมพัทธ์โดยอ้างอิงจากค่า PC ในปัจจุบันไปเป็น PC + OFFSET ค่าในเขต OFFSET จะมีค่าบวกหรือลบก็ได้ เพื่อใช้ในการกระโดดไปข้างหน้าหรือถอยหลังจากตำแหน่งปัจจุบัน

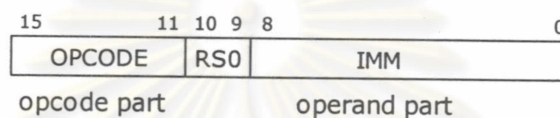
2. แบบสัมบูรณ์ (Absolute addressing mode)



รูปที่ 3.9 โครงสร้างของคำสั่งในโหมดสัมบูรณ์

เป็นแอดเดรสซิ่งโหมดสำหรับกลุ่มคำสั่งการควบคุมเกี่ยวกับการกระโดดไปยังคำสั่งเป้าหมายแบบโดยตรง โดยตำแหน่งของคำสั่งเป้าหมายจะถูกกำหนดโดยเขต ADS (Absolute address) ในคำสั่ง

3. แบบทันที (Immediate addressing mode)



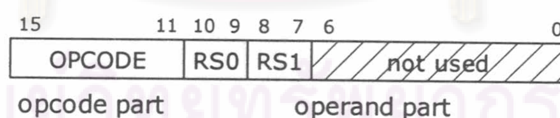
รูปที่ 3.10 โครงสร้างของคำสั่งในโหมดแบบทันที

เป็นแอดเดรสซิ่งโหมดสำหรับกลุ่มคำสั่งการคำนวณและตรรกะ มีค่าที่อยู่ในเรจิสเตอร์ที่อ้างจาก RS0 เป็นตัวตั้ง และกระทำด้วยค่าที่อยู่ในเขต IMM ผลลัพธ์ที่ได้จะนำไปเก็บลงในเรจิสเตอร์ที่ระบุในเขต RS0 ดังสมการ

$$RS0 \leftarrow RS0 \otimes IMM$$

Where ⊗ is the operation of the instruction

4. แบบเรจิสเตอร์ (Register addressing mode)



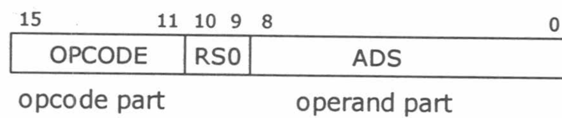
รูปที่ 3.11 โครงสร้างของคำสั่งในโหมดแบบเรจิสเตอร์

เป็นแอดเดรสซิ่งโหมดสำหรับกลุ่มคำสั่งการคำนวณและตรรกะเช่นเดียวกับแอดเดรสซิ่งโหมดแบบทันทีเพียงแต่จะเป็นการกระทำกันระหว่างเรจิสเตอร์ โดยที่จะเอาค่าในเรจิสเตอร์ที่ระบุในเขต RS0 มากระทำกับค่าในเรจิสเตอร์ในเขต RS1 แล้วนำผลลัพธ์ที่ได้ไปเก็บในเรจิสเตอร์ RS0

$$RS0 \leftarrow RS0 \otimes RS1$$

Where ⊗ is the operation of the instruction

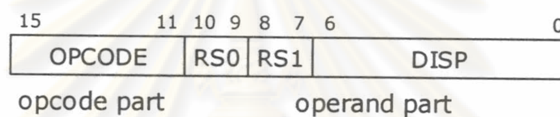
5. แบบตรง (Direct addressing mode)



รูปที่ 3.12 โครงสร้างของคำสั่งในโหมดแบบโดยตรง

เป็นแอดเดรสซิ่งโหมดสำหรับกลุ่มคำสั่งสำหรับการย้ายข้อมูลระหว่างแฟ้มเรจิสเตอร์กับหน่วยความจำข้อมูล อ้างอิงถึงข้อมูลที่อยู่ในหน่วยความจำตำแหน่งที่ระบุในเขต ADS โดยตรง ถ้าเป็นการย้ายข้อมูลจากหน่วยความจำเข้ามาในแฟ้มเรจิสเตอร์ ข้อมูลจะถูกเก็บลงในเรจิสเตอร์ที่ระบุในเขต RS0 แต่ถ้าเป็นการเขียนข้อมูลลงหน่วยความจำ จะนำเอาค่าในเรจิสเตอร์ที่ระบุในเขต RS0 ไปเก็บ

6. แบบดรรชนี (Index addressing mode)



รูปที่ 3.13 โครงสร้างของคำสั่งในโหมดแบบอ้างอิง

เป็นแอดเดรสซิ่งโหมดสำหรับกลุ่มคำสั่งการย้ายข้อมูลระหว่างเรจิสเตอร์กับหน่วยความจำข้อมูล โดยใช้ค่าในเรจิสเตอร์ที่ถูกระบุด้วยเขต RS1 เป็นฐาน บวกกับค่าที่อยู่ในเขต DISP แบบคู่ส่วนเติมเต็ม (2's complement) ถ้าเป็นการย้ายข้อมูลจากหน่วยความจำเข้ามาในเรจิสเตอร์ ข้อมูลถูกเก็บลงในเรจิสเตอร์ที่ระบุในเขต RS0 แต่ถ้าเป็นการเขียนข้อมูลลงหน่วยความจำ จะนำเอาค่าในเรจิสเตอร์ที่ระบุในเขต RS0 ไปเก็บ

7. แบบอื่นๆ นอกเหนือจากที่กล่าวมาแล้ว

มีคำสั่ง 3 คำสั่งที่ไม่อยู่ในแอดเดรสซิ่งโหมดประเภทที่กล่าวมาแล้วทั้งหมด ได้แก่คำสั่ง SC คำสั่ง SS และคำสั่ง NOP โดยคำสั่ง SC และ SS เป็นการกำหนดค่าให้กับเรจิสเตอร์พิเศษในหน่วยประมวลผล ส่วนคำสั่ง NOP เป็นคำสั่งที่ไม่มีผลการทำงานใดๆ เกิดขึ้นกับหน่วยประมวลผล รายละเอียดการทำงานของคำสั่งทั้ง 3 คำสั่งดูได้ในตารางที่ 3.6

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

ตารางที่ 3.6 รายละเอียดการทำงานของชุดคำสั่งของหน่วยประมวลผล C1

Mnemonic	Operand	Description	Operation	Addr mode
Arithmetic and logic instruction				
ADDI	RS0, IMM	Add immediate	RS0 ← RS0+IMM;	Immediate
SUBI	RS0, IMM	Subtract immediate	RS0 ← RS0-IMM;	Immediate
ANDI	RS0, IMM	Bitwise and immediate	RS0 ← RS0&IMM;	Immediate
ORI	RS0, IMM	Bitwise or immediate	RS0 ← RS0|IMM;	Immediate
XORI	RS0, IMM	Bitwise xor immediate	RS0 ← RS0^IMM;	Immediate
MOVI	RS0, IMM	Move immediate to register	RS0 ← IMM;	Immediate
CMPI	RS0, IMM	Compare the register to immediate	RS0 - IMM	Immediate
ADD	RS0, RS1	Add register	RS0 ← RS0+RS1;	Register
SUB	RS0, RS1	Subtract register	RS0 ← RS0-RS1;	Register
AND	RS0, RS1	Bitwise and register	RS0 ← RS0&RS1;	Register
OR	RS0, RS1	Bitwise or register	RS0 ← RS0|RS1;	Register
XOR	RS0, RS1	Bitwise xor register	RS0 ← RS0^RS1;	Register
ROL	RS0, RS1	Rotate left with carry	RS0 ← RS1<<1 with carry	Register
ROR	RS0, RS1	Rotate right with carry	RS0 ← RS1>>1 with carry	Register
MOV	RS0, RS1	Move register	RS0 ← RS1;	Register
CMP	RS0, RS1	Register comparison	RS0 - RS1;	Register
SC	IMM	Set the carry flag	C ← IMM[0];	Immediate
SS	IMM	Set the segment register	Segment ← IMM[7:0];	Immediate
NOP	-	No operation	No operation	Immediate
Control flow instruction				
JMP	OFFSET	Unconditional jump	PC ← PC+OFFSET;	Relative
JC	OFFSET	Jump if carry flag is set	if(C == 1) PC ← PC+OFFSET else PC ← PC+1;	Relative
JS	OFFSET	Jump if sign flag is set	if(S == 1) PC ← PC+OFFSET else PC ← PC+1;	Relative
JZ	OFFSET	Jump if zero flag is set	if(Z == 1) PC ← PC+OFFSET else PC ← PC+1;	Relative
JNC	OFFSET	Jump if carry flag is clear	if(C == 0) PC ← PC+OFFSET else PC ← PC+1;	Relative
JNS	OFFSET	Jump if sign flag is clear	if(S == 0) PC ← PC+OFFSET else PC ← PC+1;	Relative
JNZ	OFFSET	Jump if zero flag is clear	if(Z == 0) PC ← PC+OFFSET else PC ← PC+1;	Relative
CALL	ADS	Jump to subroutine	MEM[R2] ← PC; PC ← {Segment[7:4], ADS[11:0]};	Absolute

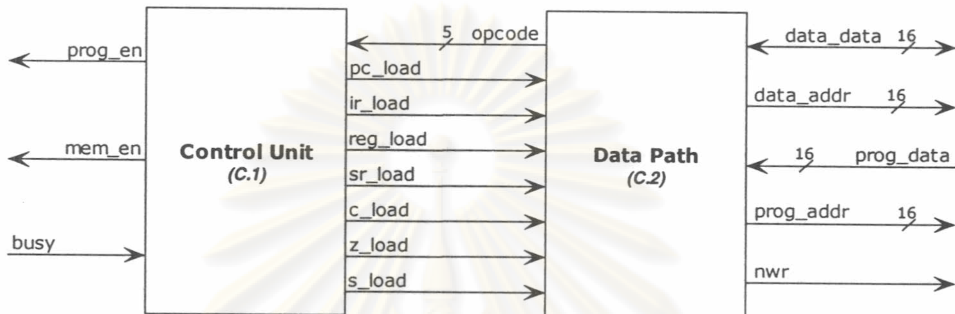
ตารางที่ 3.6 รายละเอียดการทำงานของชุดคำสั่งของหน่วยประมวลผล C1 (ต่อ)

Mnemonic	**Operand**	**Description**	**Operation**	**Addr mode**
RET	RS1, DISP	Return from subroutine	PC ← MEM[RS1+DISP];	Index
Data transfer instruction				
LDA	RS0, ADS	Load absolute address	RS0 ← MEM[ADS];	Direct
STA	RS0, ADS	Store absolute address	MEM[ADS] ← RS0;	Direct
LDW	RS0, RS1, DISP	Indexing load	RS0 ← MEM[RS1+DISP];	Index
STW	RS0, RS1, DISP	Indexing store	MEM[RS1+DISP] ← RS0;	Index

ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย

3.3.2 สถาปัตยกรรม

โครงสร้างของหน่วยประมวลผล C1 (ดูรูปที่ 1.7 อ้างอิงหน่วย ***c***) แบ่งออกเป็น 2 ส่วน ได้แก่หน่วยควบคุม (Control unit) และส่วนทางเดินข้อมูล (Data path) ดังรูปที่ 3.14 โดยหน่วยควบคุมทำหน้าควบคุมการทำงานโดยรวมของหน่วยประมวลผลตั้งแต่การอ่านคำสั่ง การแปลคำสั่ง และประมวลผลคำสั่ง โดยที่การประมวลผลคำสั่งจะเกิดจากการที่หน่วยควบคุมการทำการควบคุมส่วนทางเดินข้อมูลให้ประมวลผลข้อมูลตามการทำงานของคำสั่งนั้นๆ



รูปที่ 3.14 โครงสร้างภายในหน่วยประมวลผล C1

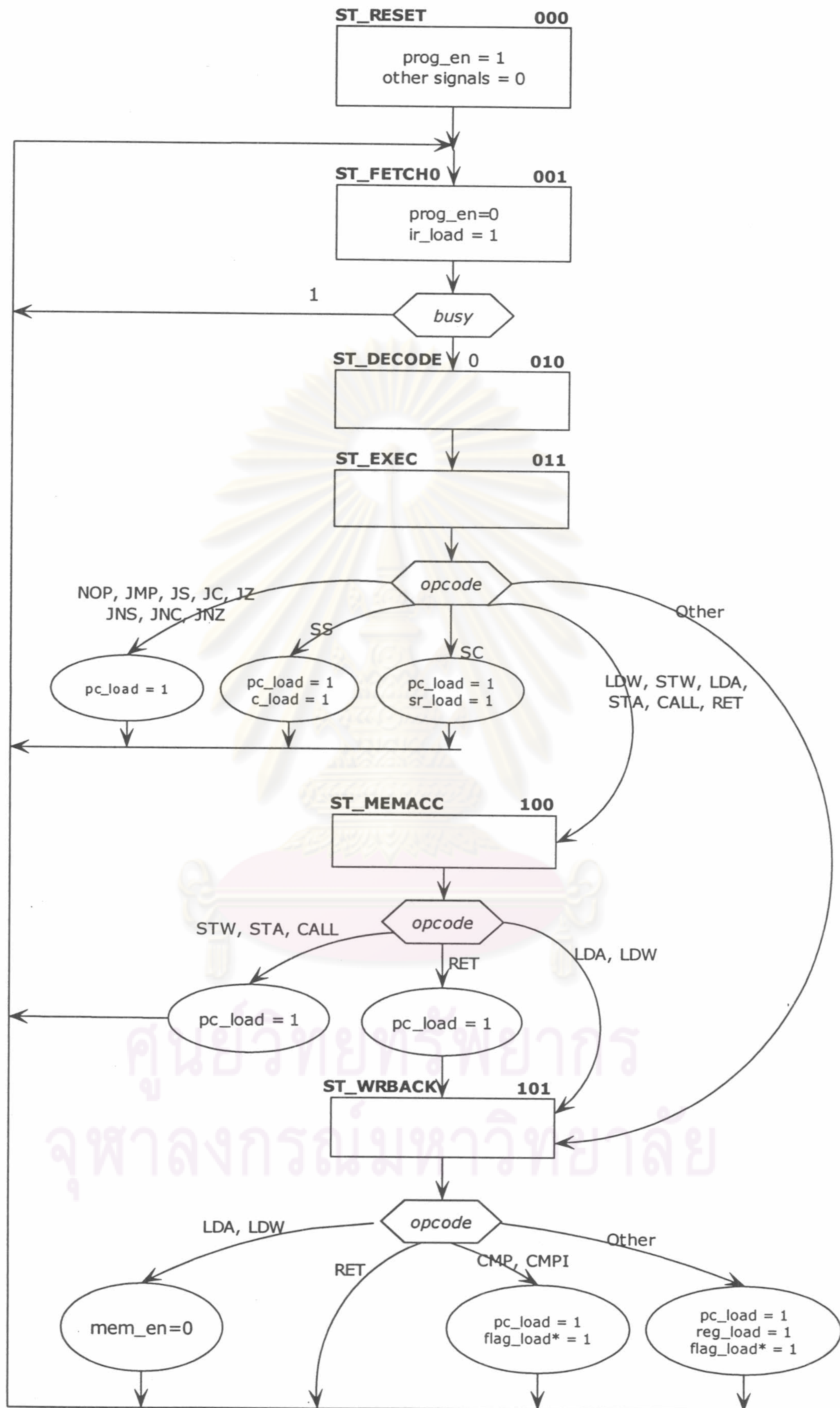
หน่วยควบคุม

หน่วยควบคุม (Control unit, ***c.1***) เป็นเครื่องสถานะจำกัด (Finite state machine) ทำหน้าที่ในการควบคุมการอ่านคำสั่ง การแปลคำสั่ง และการประมวลผล สัญญาณควบคุมต่างๆ เป็นไปดังรูปที่ 3.14 แบ่งออกเป็นสายสัญญาณที่นำไปควบคุมการอ่านคำสั่งจากหน่วยความจำโปรแกรม สัญญาณที่ใช้อ่านข้อมูลจากหน่วยความจำข้อมูล และสัญญาณที่ใช้ควบคุมการทำงานของส่วนทางเดินข้อมูล

สัญญาณ prog_en เป็นสัญญาณที่ใช้ในการควบคุมการทำงานของหน่วยความจำโปรแกรม สัญญาณ busy เป็นสัญญาณที่ใช้ในการแสดงความพร้อมของคำสั่งจากหน่วยความจำ ส่วน mem_en เป็นสัญญาณที่ใช้ในการควบคุมหน่วยความจำข้อมูล ส่วนสัญญาณที่เหลือนั้นใช้ควบคุมส่วนทางเดินข้อมูลเพื่อกระตุ้นให้เรจิสเตอร์ต่างๆ เก็บค่าเพื่อใช้ในการประมวลผลข้อมูลตามคำสั่งนั้นๆ

การเปลี่ยนแปลงของสัญญาณควบคุมเหล่านี้จะเป็นไปตามแผนภาพสถานะในรูปที่ 3.15 โดยที่สัญญาณ flag_load จะหมายถึงการให้เรจิสเตอร์แฟลกทุกตัวทำการเก็บค่าใหม่ และสัญญาณที่ไม่กล่าวถึงในแต่ละสถานะหมายถึงว่าให้สัญญาณดังกล่าวอยู่ในสถานะไม่ถูกกระตุ้น (เท่ากับ 0 ถ้าเป็นสัญญาณกระตุ้นสูง และเท่ากับ 1 สำหรับสัญญาณกระตุ้นต่ำ) และในตารางที่ตารางที่ 3.7 แสดงจำนวนสัญญาณนาฬิกาที่ใช้ในการทำงานของแต่ละคำสั่ง

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 3.15 แผนภาพสถานะของหน่วยควบคุม

ตารางที่ 3.7 สรุปจำนวนสัญญาณนาฬิกาการทำงานของแต่ละคำสั่ง

Instruction	Clock
ADDI	4
SUBI	4
ANDI	4
ORI	4
XORI	4
MOVI	4
CMPI	4
ADD	4
SUB	4
AND	4
OR	4
XOR	4
ROL	4
ROR	4
MOV	4
CMP	4
SC	3
SS	3
NOP	3
JMP	3
JC	3
JS	3
JZ	3
JNC	3
JNS	3
JNZ	3
CALL	4
LDA	5
STA	4
LDW	5
STW	4
RET	5

ส่วนทางเดินข้อมูล

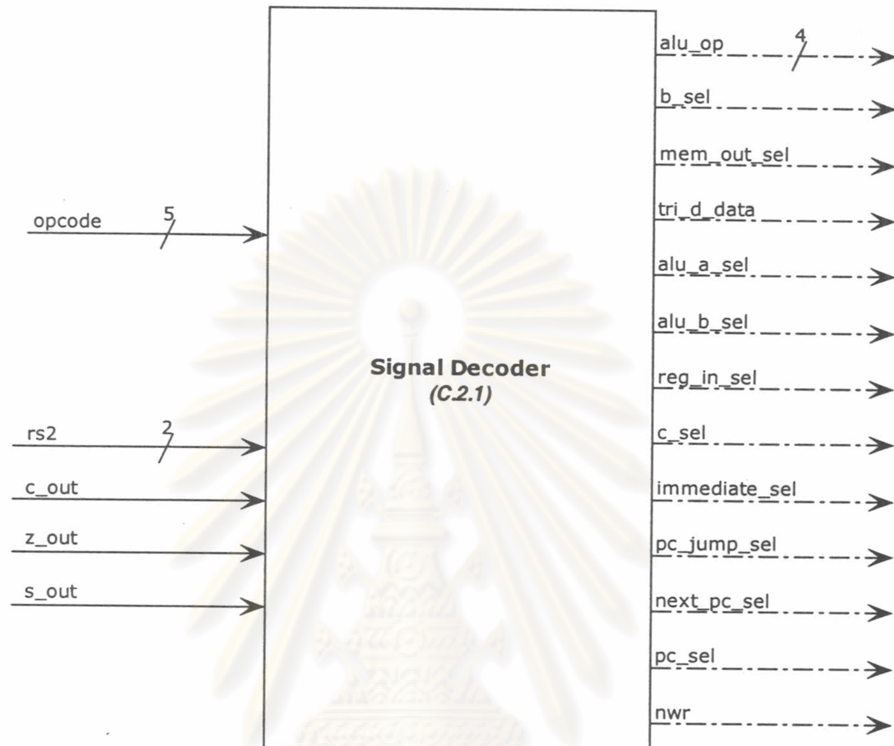
ทางเดินข้อมูล (Data path, ***c.2***) ทำหน้าที่กระทำการ (Execute) ประกอบไปด้วยวงจรเชิงผสม (Combinational circuit) และเรจิสเตอร์ที่ใช้ในการเก็บค่าที่ใช้ในการประมวลผล รายละเอียดของส่วนทางเดินข้อมูลแสดงดังรูปที่ 3.17 โดยทางเดินข้อมูลถูกควบคุมด้วยสัญญาณควบคุม[1] สองส่วนได้แก่ สัญญาณควบคุมจากหน่วยควบคุม และสัญญาณควบคุมที่ได้จากการถอดรหัสคำสั่งที่เป็นวงจรเชิงผสม (Signal Decoder, ***c.2.1***)

---

[1] จากรูปที่ 3.16 และรูปที่ 3.17 สังเกตได้ว่าสัญญาณควบคุมจะเป็นเส้นประ และบัสข้อมูลหรือสัญญาณที่ไม่ได้มาจากการถอดรหัสจะเป็นเส้นทึบ

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย

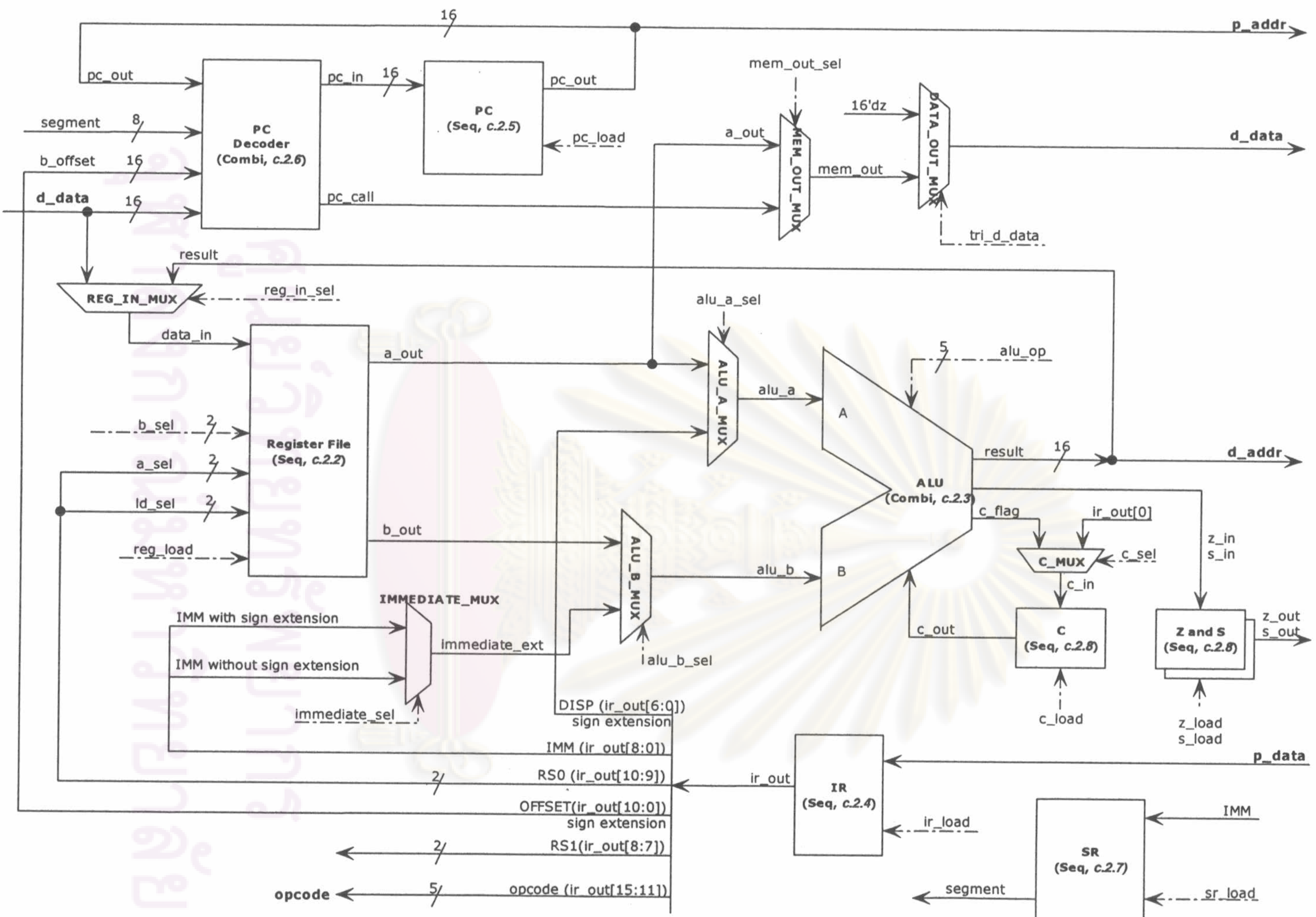
สัญญาณควบคุมที่ได้จากหน่วยควบคุมได้กล่าวไปแล้ว ส่วนสัญญาณควบคุมที่ได้จากการถอดรหัสคำสั่ง (Signal Decoder, ***c.2.1***) ที่เป็นวงจรเชิงผสมนั้นเป็นดังรูปที่ 3.16 โดยรับอินพุตเป็นรหัสดำเนินการ ค่าในเขตเรจิสเตอร์ในคำสั่ง และค่าตัวบ่งชี้ และนำมาถอดรหัสให้ได้สัญญาณควบคุม



รูปที่ 3.16 วงจรถอดรหัสคำสั่ง

ส่วนประกอบของส่วนทางเดินข้อมูลที่แสดงในรูปที่ 3.17 นั้นจะประกอบด้วยเรจิสเตอร์ IR, PC, SR, แฟ้มเรจิสเตอร์ (Register file), หน่วยคำนวณและตรรกะ (Arithmetic and Logic Unit: ALU) และวงจรถอดรหัสค่า PC (PC Decoder, ***c.2.6***) โดยในรูปส่วนประกอบที่มีสัญลักษณ์ combi หมายถึงเป็นวงจรเชิงผสม (Combinational logic) และ seq คือวงจรเชิงลำดับ (Sequential logic) ที่ทำงานตามสัญญาณนาฬิกา ซึ่งในแผนภาพจะละสัญญาณนาฬิกาซึ่งต่อกับวงจรเชิงลำดับไว้ในฐานที่เข้าใจ

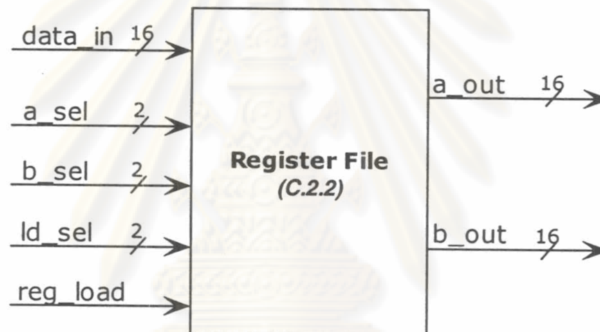
ศูนย์วิทยทรัพยากร
จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 3.17 รายละเอียดส่วนทางเดินข้อมูล

เรจิสเตอร์ IR (Instruction register, ***c.2.2***) และ PC (Program counter, ***c.2.5***) เป็นเรจิสเตอร์ขนาด 16 บิตสำหรับเก็บคำสั่งที่อ่านมาได้และเก็บตำแหน่งของคำสั่งปัจจุบันในหน่วยความจำโปรแกรมตามลำดับ ส่วนเรจิสเตอร์ SR (Segment register, ***c.2.7***) มีขนาด 8 บิต ใช้อ้างถึงส่วนของโปรแกรม (Segment) นอกเหนือจากที่หน่วยประมวลผลอ้างถึงได้ในการกระโดด (รายละเอียดอ่านในการทำงานชุดคำสั่ง)

แฟ้มเรจิสเตอร์ (Register file, ***c.2.2***) เป็นกลุ่มของเรจิสเตอร์ที่สามารถใช้งานทั่วไปได้ในระดับสถาปัตยกรรมชุดคำสั่ง มีด้วยกัน 4 เรจิสเตอร์ได้แก่ R0, R1, R2 และ R3 จากรูปที่ 3.18 แฟ้มเรจิสเตอร์สามารถให้ค่าออกของเรจิสเตอร์ออกมาได้ 2 ข้อมูลพร้อมกัน โดยเลือกผ่านสัญญาณ a_sel และ b_sel ส่วนการตั้งค่าให้กับแฟ้มเรจิสเตอร์สามารถตั้งค่าได้ครั้งละหนึ่งเรจิสเตอร์ ผ่านทางการเลือกจากสัญญาณ ld_sel โดยใช้สัญญาณ reg_load เป็นตัวกระตุ้นการตั้งค่า



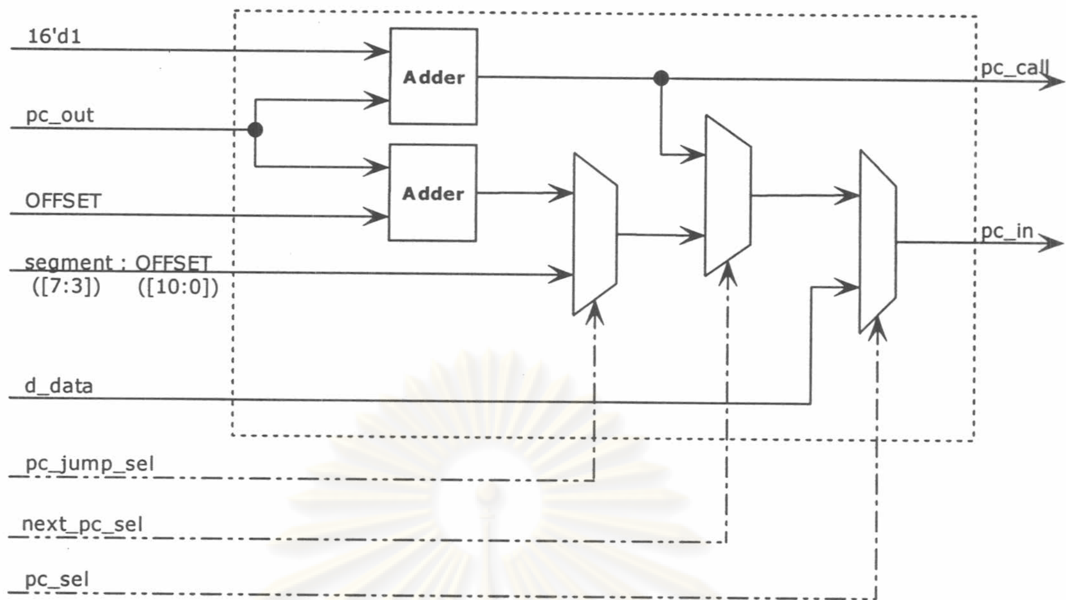
รูปที่ 3.18 แผนภาพบล็อกแฟ้มเรจิสเตอร์

หน่วยคำนวณและตรรกะ (Arithmetic and logic unit, ALU, ***c.2.3***) เป็นวงจรเชิงผสมที่ใช้ในการคำนวณค่าต่างๆ ตามตัวดำเนินการ (Operator) ที่ระบุจากสัญญาณ alu_op ขนาด 4 บิต ดังตารางที่ 3.8

ตารางที่ 3.8 การคำนวณของหน่วยคำนวณและตรรกะ

"alu_op" Encoding	Operator Name	Operation Description
0000	ADD	A + B
0001	SUB	A - B
0010	Pass A	A
0011	Pass B	B
0100	AND	A & B
0101	OR	A | B
0110	XOR	A ^ B
1010	ROL	Rotate left A with carry
1110	ROR	Rotate right A with carry

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย



รูปที่ 3.19 รายละเอียดวงจรถอดรหัสค่า PC

วงจรถอดรหัสค่า PC (PC Decoder, ***c.2.6***) เป็นวงจรเชิงผสมที่ทำหน้าที่ประเมินตำแหน่งเลขที่อยู่ถัดไปของคำสั่งที่ถูกนำมาประมวลผล ซึ่งการคำนวณมีด้วยกัน 3 ส่วนได้แก่

1. การเพิ่มขึ้นทีละ 1 เพื่ออ้างถึงคำสั่งถัดไปที่อยู่ติดกัน
2. การกระโดด ตำแหน่งถัดไปได้มาจากการบวกกันระหว่างตำแหน่งเดิมและค่าจากเขต OFFSET ในคำสั่ง
3. การกระโดดไปทำงานในโปรแกรมย่อยและการเรียกกลับจากโปรแกรมย่อย

ตำแหน่งเลขที่อยู่ถัดไปที่จะนำไปเก็บในเรจิสเตอร์ PC จะถูกเลือกตามการถอดรหัสที่ได้จากการถอดรหัสคำสั่งจากวงจรถอดรหัสสัญญาณควบคุม (***c.2.1***) โดยใช้มัลติเพล็กเซอร์ 2 ต่อ 1 ขนาด 16 บิตจำนวน 3 ตัว

สุดท้ายเรจิสเตอร์ SR เป็นเรจิสเตอร์ที่ใช้ในการเก็บค่าเซกเมนต์ (Segment) เพื่อใช้ในการกระโดดไปทำงานยังโปรแกรมย่อยที่อยู่นอกขอบเขตที่สามารถระบุได้ในระดับชุดคำสั่งหรืออยู่ในตำแหน่งที่เกิน $2^{11} - 1$ โดยค่าในเรจิสเตอร์ SR ใช้ร่วมกับตัวถูกดำเนินการ ADS ในคำสั่ง CALL ผลจากการกระโดดด้วยคำสั่งนี้จะทำให้ค่าเรจิสเตอร์ PC มีค่าเปลี่ยนไปดังนี้

```
PC ← {SR[7:3], ADS[10:0]}
```