241-302 Computer Engineering Lab II ภาคเรียนที่ 2 ปีการศึกษา 2554

Lab 3HB05 การใช้งานภาษาแอสเซมบลี้และภาษาซีกับซีพียู AVR
ผู้สอน ดร. ปัญญยศ ไชยกาฬ

1. วัตถุประสงค์

- เพื่อให้นักศึกษาได้เรียนรู้วิธีการเขียนโปรแกรมประยุกต์ใช้งานไมโครคอนโทรลเลอร์ AVR ด้วยภาษาแอสเซมบลี้และภาษาซี
- เพื่อให้นักศึกษาได้เรียนรู้เทคนิคการดีบักโปรแกรมภาษาแอสเซมบลี้ และภาษาซีก่อนที่จะนำโปรแกรมลงทดสอบบนฮาร์ดแวร์จริง
- เพื่อให้นักศึกษาทำการทดลองต่อวงจรเชื่อมต่อภายนอกกับบอร์ดไมโครคอนโทรลเลอร์ AVR และทดสอบการทำงานของโปรแกรมที่เขียนขึ้นบนฮาร์ดแวร์จริงได้

2. เป้าหมาย

- นักศึกษาสามารถนำโปรแกรมควบคุมซีพียู AVR ที่เขียนขึ้นมาทำการคอมไพล์และอัพโหลดโปรแกรมลงบอร์ดทดลองได้
- นักศึกษาสามารถใช้โปรแกรม AVRStudio ในการดีบักโปรแกรมภาษาแอสเซมบลี้และภาษาซีของซีพียู AVR ได้อย่างมีประสิทธิภาพ
- นักศึกษาสามารถต่อวงจรเชื่อมต่อภายนอกกับบอร์ดไมโครคอนโทรลเลอร์ AVR และทดสอบการทำงานของโปรแกรมบนฮาร์ดแวร์จริงได้

3. กำหนดส่งงานและวิธีการส่งงาน

- *วิธีการตรวจ Checkpoint*
  - ในCheckpoint 1 นั้นให้นักศึกษาทำ Checkpoint 1.1-1.3 ให้เสร็จก่อน แล้วจึงค่อยเรียกผู้คุมแลบตรวจทั้ง 3 ข้อย่อยในคราวเดียว
  - ใน Checkpoint 2 ให้ทำ Checkpoint 2.1 เสร็จแล้วจึงเรียกผู้คุมแลบตรวจ จากนั้นจึงทำ Checkpoint 2.2-2.3 ให้เสร็จแล้วจึงเรียกผู้คุมแลบมาตรวจ
- *การส่ง Logbook*
  ให้ส่งหลังจากทำแลบ 3HB08 เสร็จแล้ว
- *การส่งคำถามท้ายการทดลอง*
  ให้ส่งใน LMS ภายในวันที่อาจารย์ประกาศ (จะแจ้งให้ทราบในวันทำแลบ)
- *การสอบ*
  กำหนดสอบ หลังจากทำแลบเสร็จแล้ว 2 สัปดาห์ (จะแจ้งวันเวลาสอบให้ทราบในภายหลัง)

4. การให้สัดส่วนคะแนน

คะแนนของการทดลองนี้แบ่งออกเป็น 4 ส่วน ได้แก่

- Checkpoint 30 %
- Logbook 15 %
- คำถามท้ายการทดลอง 15 %
- สอบปฏิบัติการ 40 %

5. วัสดุอุปกรณ์ที่ใช้ในการทดลอง

- สายไฟแข็งสำหรับต่อทดลองวงจรบน Breadboard 20 เส้น
- ชุดทดลองดิจิตอล (Logic Trainer) 1 ชุด
- สาย USB 1 เส้น
- บอร์ดไมโครคอนโทรลเลอร์ AVR รุ่น Diecilima ATMega32 1 บอร์ด
- แอลอีดีแบบ 7 เซกเมนต์ชนิด Common Cathode 1 ตัว
- ดิปสวิทช์ 8 บิต 1 ตัว
- ตัวต้านทาน 220 โอห์ม 8 ตัว
- ตัวต้านทาน 10 กิโลโอห์ม 8 ตัว

6. แนะนำซีพียู AVR

ไมโครคอนโทรลเลอร์ เป็นหน่วยประมวลผลขนาดเล็ก ซึ่งรวมเอาความสามารถของไมโครโพรเซสเซอร์กับอุปกรณ์เชื่อมต่อภายนอก เช่นหน่วยความจำ อุปกรณ์อินพุตเอาต์พุต วงจรกำเนิดสัญญาณนาฬิกา เข้ามารวมอยู่ภายในไอซีชิพเพียงตัวเดียว ส่งผลให้ระบบควบคุมซึ่งใช้ไมโครคอนโทรลเลอร์เป็นหน่วยประมวลผลมีขนาดของวงจรควบคุมเล็กและสะดวกต่อการใช้งาน

แม้ว่าระบบไมโครคอนโทรลเลอร์จะมีหลายๆ อุปกรณ์อยู่ภายในไอซีชิพตัวเดียว แต่ก็มีความสามารถจำกัด เมื่อเทียบกับระบบซึ่งใช้ไมโครโพรเซสเซอร์ ยกตัวอย่างเช่น หน่วยความจำ จะมีให้ใช้เพียงไม่มากนัก ดังนั้นระบบไมโครคอนโทรลเลอร์จะเหมาะสมสำหรับการควบคุมงานที่ไม่ซับซ้อนและต้องการความสามารถในการประมวลผลที่ไม่สูงมากนัก ยกตัวอย่างเช่นในระบบ Embedded system หรือระบบสมองกลฝังตัว เช่น เครื่องซักผ้า เตาอบไมโครเวฟ เครื่องปรับอากาศ เป็นต้น

สถาปัตยกรรมของไมโครคอนโทรลเลอร์ที่ใช้งานอยู่ในปัจจุบันมีให้เลือกใช้งานมากมายหลายตัว ยกตัวอย่างเช่น

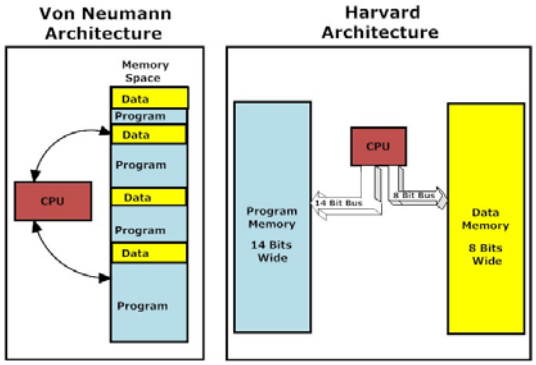
- MCS-51 (8-bit) ออกแบบโดย intel
- MCS-96 (16-bit) ออกแบบโดย intel
- PIC ออกแบบโดย Microchip Technology
- AVR ออกแบบโดย Atmel
- ARM ออกแบบโดย ARM Holdings
- 68HC11 ออกแบบโดย Motorola
- Rabbit 2000 ออกแบบโดย Rabbit Semiconductor

ในอดีต ไมโครคอนโทรลเลอร์ที่นิยมใช้งานกันคือตระกูล MCS-51 แต่ในปัจจุบันความต้องการความสามารถในการประมวลผลมีเพิ่มขึ้น ทำให้มีสถาปัตยกรรมอื่นๆ ที่มีประสิทธิภาพสูงกว่าเริ่มได้รับความนิยมในการใช้งานขึ้นมาแทนในแลบนี้จะสอนให้นักศึกษาได้เรียนรู้สถาปัตยกรรม AVR เนื่องจากมีประสิทธิภาพที่สูง และมีราคาถูกกว่าสถาปัตยกรรมอื่นในราคาที่ใกล้เคียงกัน นอกจากนี้ยังมีข้อดีคือมี ฟรีแวร์หลายตัวให้ใช้งานสำหรับพัฒนาระบบได้ทั้งภาษาระดับต่ำคือภาษาแอสเซมบลี้และภาษาระดับสูงเช่น ภาษาซี เป็นต้น

สถาปัตยกรรมของ AVR แบ่งออกเป็น 2 ตระกูลคือ

- 8-bit AVR
- 32-bit AVR

ในแลบนี้จะกล่าวถึงสถาปัตยกรรมของ AVR ขนาด 8 บิตเท่านั้น โดยในสถาปัตยกรรม AVR ออกแบบโดย ATMEL เป็นซีพียูแบบ RISC (Reduced Instruction Set Computer) มีสถาปัตยกรรมการต่อหน่วยความจำแบบ Harvard ซึ่งแยกหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลออกจากกันโดยเด็ดขาด ดังแสดงในรูปที่ 1.1 โดยใช้หน่วยความจำแบบ Flash สำหรับเป็นหน่วยความจำโปรแกรม และใช้หน่วยความจำแบบ SRAM สำหรับหน่วยความจำข้อมูล และนอกจากนี้ยังมีหน่วยความจำแบบ EEPROM ซึ่งสามารถเก็บข้อมูลเอาไว้ได้โดยไม่จำเป็นต้องมีไฟเลี้ยงอีกด้วย



รูปที่ 1.1 เปรียบเทียบการจัดการหน่วยความจำของสถาปัตยกรรมแบบ Von-Neumann และ Harvard

จากรูปที่ 1.1 จะเห็นว่าโพรเซสเซอร์ที่ใช้สถาปัตยกรรมแบบ Harvard จะแยกหน่วยความจำสำหรับเก็บข้อมูลออกจากโปรแกรมอย่างชัดเจน สถาปัตยกรรม AVR และ MCS-51 จะใช้รูปแบบนี้ในการจัดการหน่วยความจำ ส่วนสถาปัตยกรรมแบบ Von-neumann การตัดสินใจว่าจะเก็บโปรแกรมหรือข้อมูลจะแบ่งเก็บอย่างไรจะทำได้อย่างอิสระ โดยขึ้นอยู่กับโปรแกรมเมอร์ หรืออาจจะเป็นระบบปฏิบัติการเป็นผู้ดำเนินการให้

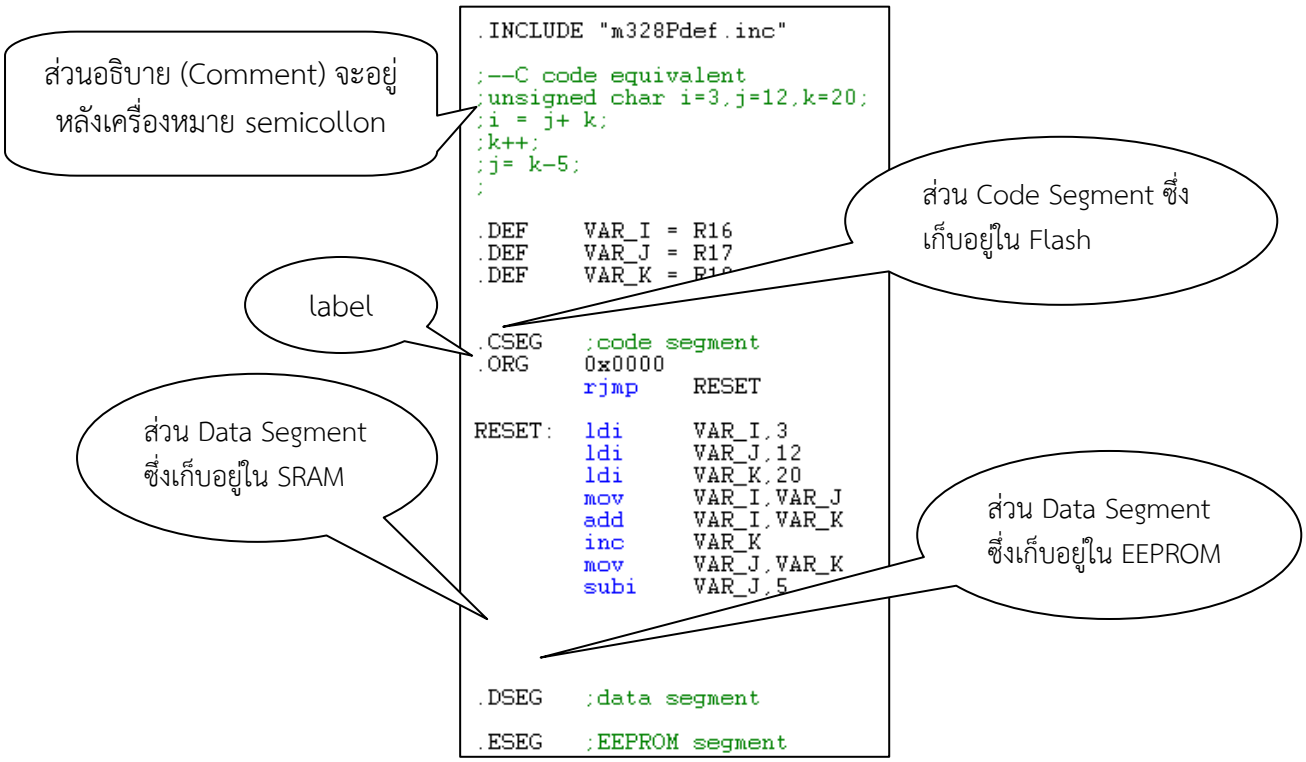
ลักษณะเด่นของสถาปัตยกรรม AVR คือ คำสั่งส่วนใหญ่สามารถทำงานได้เสร็จภายใน 1 clock cycle ตัวซีพียู AVR ขนาด 8 บิตจะแบ่งออกเป็นประเภทการใช้งานได้ 5 กลุ่ม ได้แก่

- tinyAVR เป็นซีพียูในรุ่นเล็ก ซึ่งต้องการความเล็กกะทัดรัดของวงจร โดยเหมาะกับระบบควบคุมขนาดเล็กๆ ที่ต้องการหน่วยความจำและวงจรสนับสนุนไม่มากนัก ซีพียูในรุ่นนี้จะมีราคาถูกกว่ากลุ่มอื่น
- megaAVR จะมีชื่ออีกอย่างว่า ATmega โดยมีวงจรสนับสนุนภายในเพิ่มเติมตลอดจนเพิ่มขนาดของหน่วยความจำให้ใช้งานมากกว่าตระกูล Tiny เหมาะกับงานควบคุมทั่วๆ ไป
- XMEGA เพิ่มความละเอียดของวงจร A/D จากปกติมีความละเอียด 10 บิตในรุ่นเล็กกว่าเป็น 12 บิต และวงจร DMA controller ซึ่งช่วยลดภาระของซีพียูในการควบคุมการรับส่งข้อมูลระหว่างอุปกรณ์ I/O กับหน่วยความจำ
- FPSLIC (AVR core with FPGA) สำหรับงานที่ต้องการควบคุมที่ต้องการความยืดหยุ่นในขั้นตอนการออกแบบและพัฒนา โดยผู้ออกแบบสามารถออกแบบวงจรในระดับฮาร์ดแวร์เพิ่ม เติมด้วยภาษาบรรยายฮาร์ดแวร์ (HDL: Hardware Description Language) เช่น ภาษา VHDL หรือภาษา Verilog และให้วงจรที่ออกแบบทำงานร่วมกับซีพียู AVR core
- Application Specific AVR เป็นซีพียูที่ออกแบบมาโดยเพิ่มวงจรควบคุมเฉพาะด้านเข้าไปซึ่งไม่พบในซีพียูกลุ่มอื่นๆ เช่นวงจร USB controller หรือวงจร CAN bus เป็นต้น

ซีพียู AVR มีให้เลือกใช้งานหลายเบอร์ แต่ละเบอร์จะมีขนาด ราคา ความสามารถ และขนาดหน่วยความจำตลอดจนถึงวงจรสนับสนุนภายในที่แตกต่างกันออกไป ในหัวข้อแลบ ปี 3 เทอม 2 นี้ จะเลือกใช้ซีพียูรุ่น ATmega32-16PU ให้นักศึกษาใช้เป็นหลัก ซึ่งมีคุณสมบัติดังนี้

- หน่วยความจำโปรแกรมแบบ FLASH ขนาด 32 กิโลไบต์
- หน่วยความจำข้อมูลแบบ SRAM ขนาด 2 กิโลไบต์
- หน่วยความจำข้อมูลแบบ EEPROM ขนาด 1 กิโลไบต์
- สนับสนุนการเชื่อมต่อแบบ I^2C bus
- พอร์ตอินพุตเอาต์พุตจำนวน 4 พอร์ต (พอร์ตละ 8 บิต รวม 32 บิต)
- วงจรสื่อสารอนุกรม
- วงจรนับ/จับเวลาขนาด 8 บิต จำนวน 2 ตัว และขนาด 16 บิตจำนวน 1 ตัว
- สนับสนุนช่องสัญญาณสำหรับสร้าง Pulse Width Modulation (PWM) จำนวน 6 ช่องสัญญาณ
- วงจรแปลงอนาลอกเป็นดิจิตอลขนาด 10 บิตในตัวจำนวน 8 ช่อง
- ความถี่ใช้งานสูงสุด 16 MHz

รูปที่ 1.2 ซีพียู ATMEGA32 แบบตัวถัง PDIP ขนาด 40 ขา



รูปที่ 1.3 โครงสร้างภาษาแอสเซมบลี้ของสถาปัตยกรรม AVR

6.1 แนะนำภาษาแอสเซมบลี้ของ AVR

โครงสร้างภาษาแอสเซมบลี้ของ AVR แสดงให้เห็นดังรูปที่ 1.1 ซึ่งจะเห็นว่ามีความแตกต่างจากภาษาแอสเซมบลี้ของสถาปัตยกรรมอื่นๆ ไม่มากนัก ในการเขียนโปรแกรมภาษาแอสเซมบลี้ เราจะต้องทำการใช้ชุดคำสั่งของซีพียูในการเข้าถึงหน่วยความจำและข้อมูลในรีจิสเตอร์โดยตรง ส่งผลให้ภาษาแอสเซมบลี้มีความยุ่งยากในการใช้งานมากกว่าระดับสูงทั่วไป อย่างไรก็ดีภาษาแอสเซมบลี้มีข้อดีกว่าภาษาอื่นตรงที่ขนาดของโปรแกรมมีขนาดเล็กมาก และมีความเร็วในการทำงานที่สูงกว่าภาษาอื่นๆ การเรียนรู้ภาษาแอสซมบลี้ ช่วยให้นักศึกษาสามารถเข้าใจการทำงานภายในของไมโครโพรเซสเซอร์ได้เป็นอย่างดี การเข้าใจภาษาแอสเซมบลี้จะช่วยให้นักศึกษาสามารถที่จะดีบัก (Debug) เพื่อทำการตรวจสอบการทำงานของโปรแกรมในกรณีที่โปรแกรมที่เขียนขึ้นด้วยภาษาระดับสูงมีปัญหาได้ สำหรับรายละเอียดเชิงลึกในการเขียนโปรแกรมภาษาแอสเซมบลี้ขอให้นักศึกษาได้ศึกษาจากเอกสาร AVR Assembler User Guide ในเอกสารแลบนี้จะทำการยกตัวอย่างการเขียนภาษาแอสเซมบลี้อย่างง่ายเพื่อเป็นพื้นฐานในการนำซีพียูไปประยุกต์ใช้งานระดับสูงขึ้นในโอกาสต่อไป

การเขียนโปรแกรมภาษาแอสเซมบลี้ จะต้องทำการเข้าถึงรีจิสเตอร์ของซีพียูโดยตรง ซีพียู AVR มีรีจิสเตอร์ใช้งานทั่วไปจำนวน 32 ตัวคือ R0-R31 ในการเขียนโปรแกรม แนะนำให้นักศึกษาใช้งานรีจิสเตอร์ตั้งแต่ R16 เป็นต้นไป นอกจากนี้ รีจิสเตอร์ R26-R31 ยังสามารถนำมาทำเป็นรีจิสเตอร์ขนาด 16 บิต ได้จำนวน 3 ตัวคือ รีจิสเตอร์ X, Y และ Z รูปที่ 1.4 และ 1.5 แสดงชุดคำสั่งของสถาปัตยกรรม AVR

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

รูปที่ 1.4 ชุดคำสั่งของ AVR

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0...6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3...0)←Rd(7...4),Rd(7...4)←Rd(3...0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

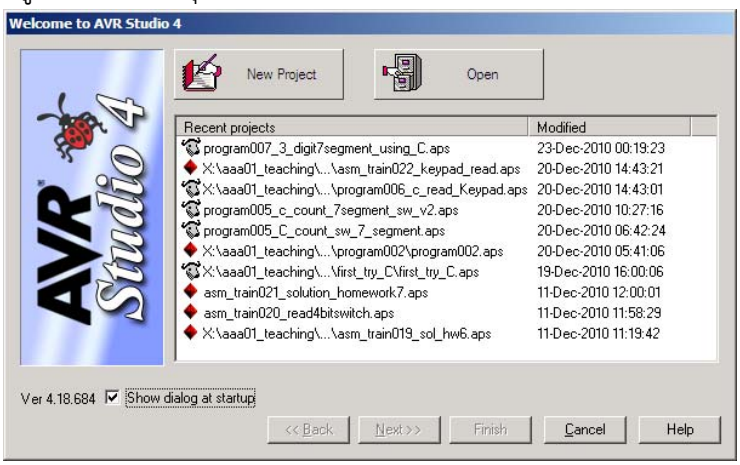
รูปที่ 1.5 ชุดคำสั่งของ AVR (ต่อ)

6.2 แนะนำซอฟต์แวร์ AVRStudio

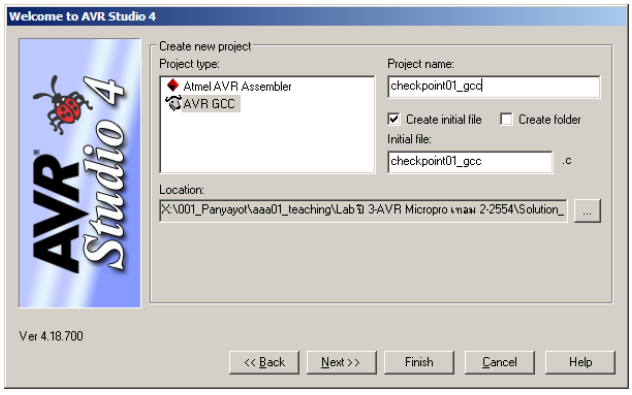
โปรแกรม AVRStudio เป็นซอฟต์แวร์ที่พัฒนาโดย ATMEL ซึ่งแจกจ่ายให้ใช้งานได้ฟรี ซึ่งใช้เป็นสภาพแวดล้อมในการพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี้หรือภาษาซีก็ได้ ในแลบนี้จะยกตัวอย่างการใช้ซอฟต์แวร์ AVRStudio ในการพัฒนาโปรแกรมภาษาแอสเซมบลี้ และภาษาซี

เรียกใช้งานโปรแกรมจาก Start menu->All Programs->ATmel AVR Tools->AVR Studio 4

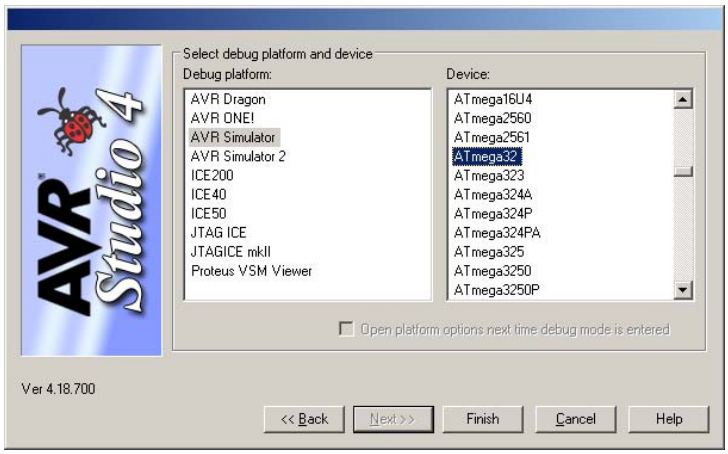
โปรแกรมจะแสดงหน้าต่าง ดังรูป ให้เลือกกดปุ่ม New Project



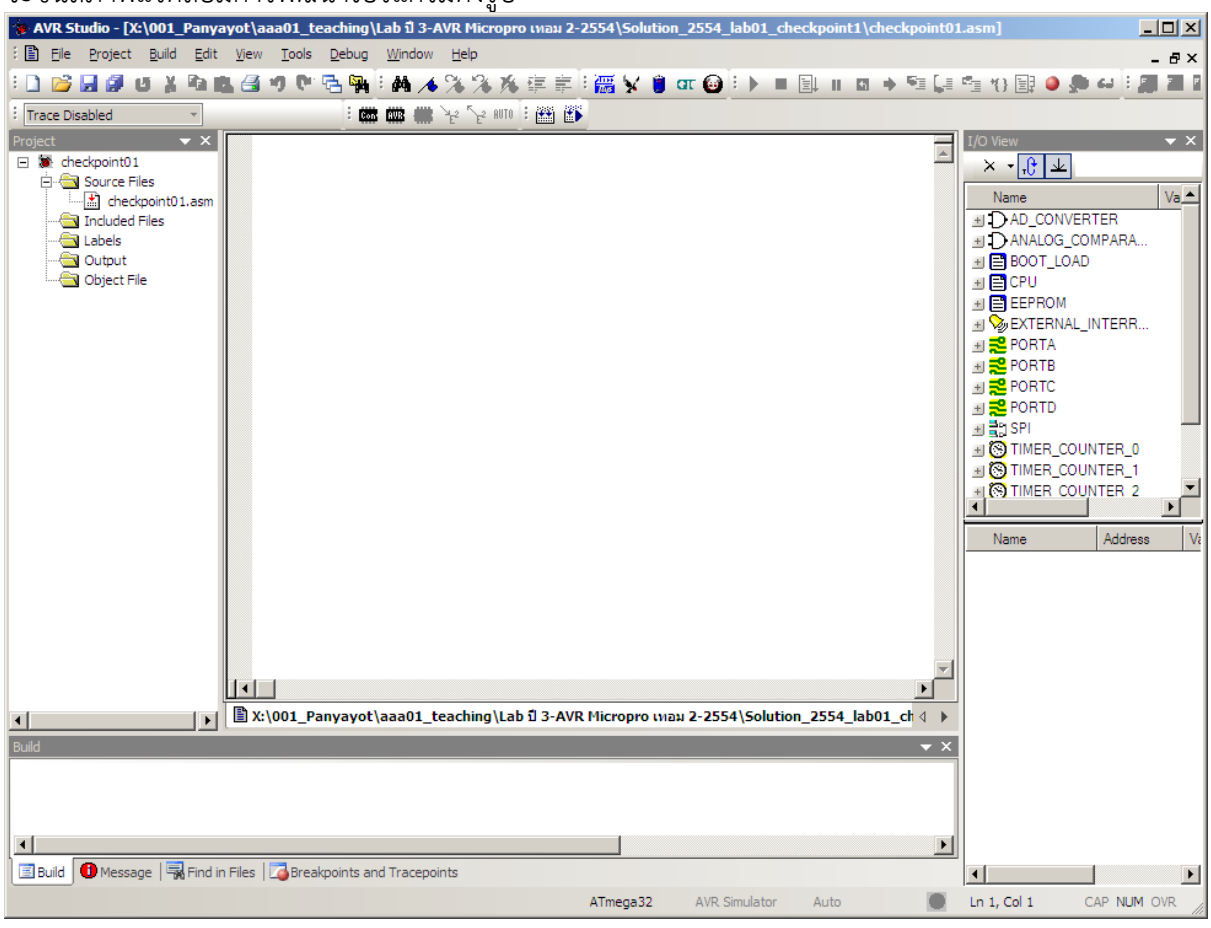
เลือกสร้างโปรเจกต์ด้วยภาษาแอสเซมบลี้ดังรูปซ้ายมือ หรือเลือกพัฒนาโปรเจกต์ด้วยภาษาซีดังรูปด้านขวามือ โดยป้อนชื่อโปรเจกต์ลงไป



เลือกซีพียูรุ่น ATmega32 และสภาพแวดล้อม AVR Simulator จากนั้นกดปุ่ม Finish



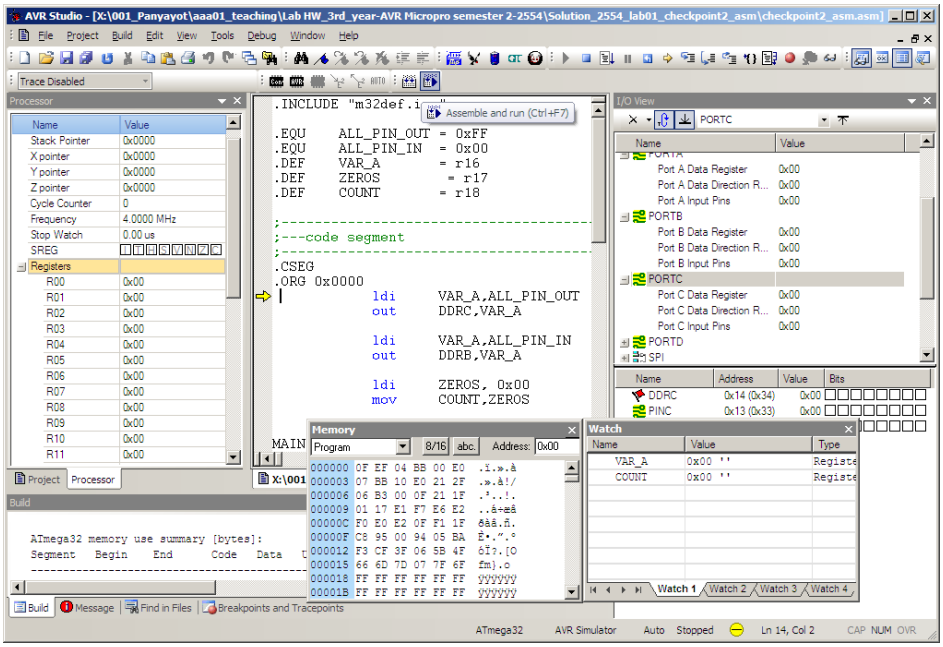
จะขึ้นสภาพแวดล้อมการพัฒนาโปรแกรมดังรูป



6.3 การใช้โปรแกรม AVRStudio ในการดีบักโปรแกรมภาษาแอสเซมบลี้

การดีบักโปรแกรม เป็นขั้นตอนสำคัญอย่างหนึ่งในการพัฒนาซอฟต์แวร์ หากอัลกอริทึมที่ต้องการเขียนมีความซับซ้อน อาจทำให้การทำงานของโปรแกรมออกมาไม่ได้อย่างที่เราคาดหวังไว้ในตอนเริ่มแรก ขั้นตอนการดีบักจะช่วยให้เราสามารถที่จะไล่การทำงานของโปรแกรมในแต่ละส่วน เพื่อที่จะค้นหาส่วนที่ผิดพลาดในโปรแกรมได้ง่ายขึ้น ในการดีบักโปรแกรมภาษาแอสเซมบลี้นั้น โปรแกรม AVRStudio ช่วยให้เราสามารถติดตามการเปลี่ยนแปลงของรีจิสเตอร์ได้ทีละคำสั่ง ผู้ทดสอบโปรแกรมสามารถที่จะทดลองป้อนค่าอินพุตพอร์ตให้โปรแกรมเพื่อที่จะดูการตอบสนองของซีพียูในการรันโปรแกรมแต่ละคำสั่งได้ ขั้นตอนการดีบักโปรแกรมภาษาแอสเซมบลี้ด้วย AVRStudio มีดังนี้

- เข้าเมนู Debug เลือก Start Debugging โปรแกรม AVRStudio จะทำการเลื่อนรีจิสเตอร์ Program Counter มาชี้ที่คำสั่งแรกเพื่อเตรียมพร้อมสำหรับการเอกซีคิวต์คำสั่ง ผู้พัฒนาโปรแกรมสามารถที่จะสั่งให้ซีพียูเอกซีคิวต์ทีละคำสั่งด้วยการกดปุ่ม F10
- ด้านซ้ายของโปรแกรม จะแสดงค่าในรีจิสเตอร์ R0-R31 รีจิสเตอร์ X, Y, Z และรีจิสเตอร์ Stack pointer รวมทั้ง Status Register ซึ่งผู้พัฒนาโปรแกรมสามารถตรวจสอบการเปลี่ยนของค่าในรีจิสเตอร์ดังกล่าวได้ทีละคำสั่ง ดังรูปที่ 1.6
- ด้านขวาของโปรแกรมแสดงค่าใน Peripheral ต่างๆ ของซีพียูไม่ว่าจะเป็น วงจรพอร์ตขนาน วงจรนับ/จับเวลา วงจรแปลงค่าสัญญาณอนาลอกเป็นดิจิตอล และวงจรอื่นๆ ซึ่งผู้พัฒนาโปรแกรมสามารถที่จะทดลองป้อนค่าให้กับตัว Peripheral ต่างๆ เพื่อตรวจสอบการตอบสนองของซีพียูที่จะรันคำสั่งสำหรับนำค่าจาก Peripheral เหล่านี้มาใช้งาน
- ผู้พัฒนาโปรแกรมสามารถตรวจดูข้อมูลในหน่วยความจำโปรแกรม หน่วยความจำข้อมูลได้ด้วยการเลือกเมนู View>Memory
- ผู้พัฒนาโปรแกรมสามารถตรวจดูข้อมูลในตัวแปรที่ตั้งเอาไว้ได้ ด้วยการเลือกเมนู View>Watch ซึ่งเปิดโอกาสให้ผู้พัฒนาสามารถ Add ตัวแปรที่ต้องการติดตามค่าสถานะไว้ใน Watch Window ได้ ดังรูปที่ 1.6

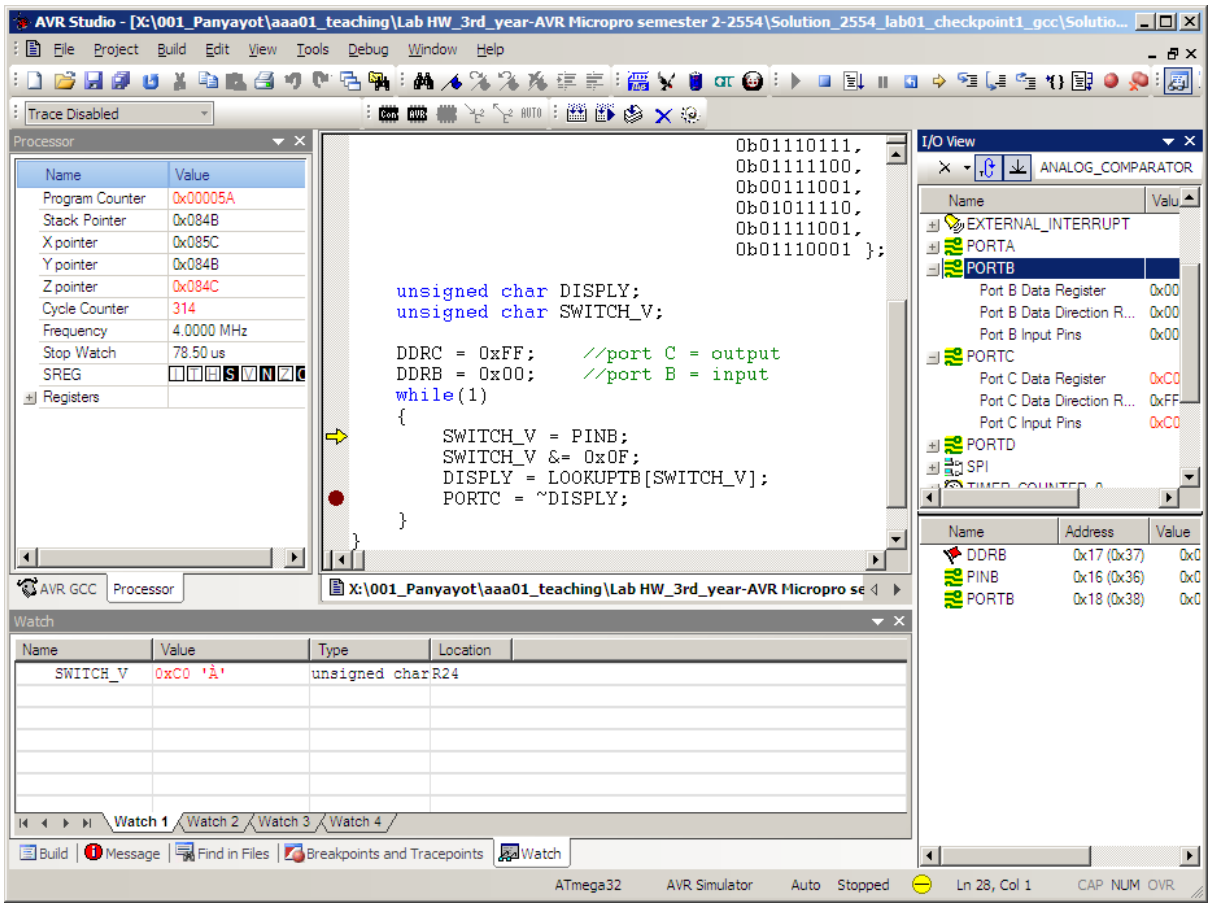


รูปที่ 1.6 หน้าจอการดีบักโปรแกรมภาษาแอสเซมบลี้ของ AVRStudio

6.4 การใช้โปรแกรม AVRStudio ในการดีบักโปรแกรมภาษาซี

การดีบักโปรแกรมภาษาซีของซอฟต์แวร์ AVRStudio จะคล้ายกับการดีบักโปรแกรมภาษาซีทั่วไปซึ่งนักศึกษาคุ้นเคยอยู่แล้วบนคอมไพเลอร์ของเครื่อง PC ไม่ว่าจะเป็นซอฟต์แวร์ SDCC หรือ Microsoft Visual C++ หรือคอมไพเลอร์ตัวอื่นๆ โดยผู้เขียนโปรแกรมสามารถที่จะติดตามการทำงานของโปรแกรมได้ครั้งละ 1 คำสั่ง ขั้นตอนการดีบักโปรแกรมภาษาซีด้วย AVRStudio มีดังนี้

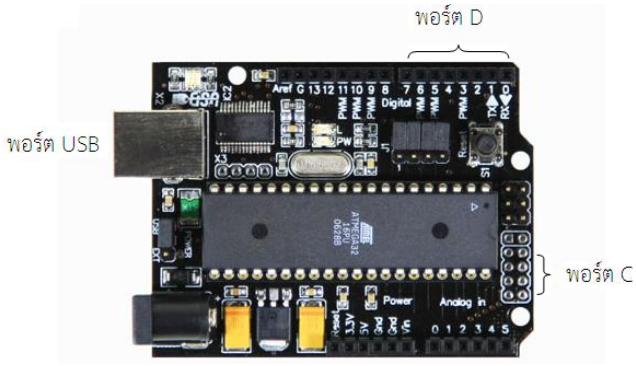
- เข้าเมนู Debug เลือก Start Debugging โปรแกรม AVRStudio จะทำการเลื่อน Program Counter มาชี้ที่คำสั่งแรกเพื่อเตรียมพร้อมสำหรับการเอกซีคิวต์คำสั่ง ผู้พัฒนาโปรแกรมสามารถที่จะสั่งให้ซีพียูเอกซีคิวต์ทีละคำสั่งด้วยการกดปุ่ม F10
- ผู้ดีบักโปรแกรมสามารถที่จะเข้าไปไล่การทำงานของคำสั่งในฟังก์ชั่นย่อยได้ด้วยการเลือกเมนู Debug>Step into หรือกดปุ่ม F11
- หากผู้ดีบักโปรแกรมไม่ต้องการเข้าไปดูรายละเอียดการทำงานของฟังก์ชั่นย่อยก็สามารถที่จะเลือกเมนู Debug>Step Over หรือกดปุ่ม F10 เพื่อข้ามการทำงานในส่วนของรายละเอียดของฟังก์ชั่นย่อยนั้นได้



รูปที่ 1.7 หน้าจอการดีบักโปรแกรมภาษาซีของ AVRStudio

6.5 บอร์ด Diecimila ATMega32

เป็นบอร์ดทดลองซึ่งนักศึกษาจะต้องใช้ในการทำแลบฮาร์ดแวร์ทั้ง 4 แลบในภาคการศึกษานี้ ตัวบอร์ดมีไอซี FT232 ทำหน้าที่จำลองพอร์ตอนุกรมชนิด RS-232 ขึ้นมาบนบัสยูเอสบีช่วยให้ผู้ใช้สามารถติดต่อกับบอร์ดผ่านบัสยูเอสบีได้ ซีพียูบนบอร์ดคือ ATMega32 ซึ่งเป็นไอซีตัวถัง DIP ขนาด 40 ขา ประกอบด้วยพอร์ตใช้งาน ดังแสดงให้เห็นในรูปที่ 1.8



รูปที่ 1.8บอร์ด Diecimila ATMega32

6.6 การใช้โปรแกรม Arduino ในการอัพโหลดโปรแกรมลงบนบอร์ดไมโครคอนโทรลเลอร์ AVR

เมื่อผู้พัฒนาโปรแกรมทำการทดสอบความถูกต้องของอัลกอริทึมที่ออกแบบเสร็จแล้วก็จะต้องทำการอัพโหลดไฟล์ภาษาเครื่องที่ได้ลงสู่ตัวไมโครคอนโทรลเลอร์ซึ่งในการทดลองนี้จะใช้โปรแกรม Arduino ซึ่งมีความสามารถในการคอมไพล์ซอร์สโค้ดภาษาซีรวมอยู่ด้วย อย่างไรก็ดี ตัวโปรแกรม Arduino ยังไม่มีขีดความสามารถในการดีบักโปรแกรม ดังนั้น ขั้นตอนการพัฒนาซอฟต์แวร์จึงต้องใช้งานควบคู่กับโปรแกรม AVRStudio ซึ่งได้กล่าวถึงในหัวข้อที่ผ่านมา ในการอัพโหลดโปรแกรมลงบอร์ดไมโครคอนโทรลเลอร์ด้วยโปรแกรม Arduino มีขั้นตอนดังต่อไปนี้

- ต่อสาย USB เชื่อมระหว่างเครื่อง PC กับบอร์ด Diecimila ATMega32
- เปิดโปรแกรม Arduino Version 022 ขึ้นมา
- เลือกเมนู Tools>Board>ArduinoMega32 เพื่อตั้งค่าชนิดของบอร์ดให้ตรงกับฮาร์ดแวร์ที่ใช้ในการทดลอง
- ก๊อปปี้ซอร์สโค้ดภาษาซีซึ่งได้ทดสอบความถูกต้องในการทำงานด้วย AVRStudio เสร็จเรียบร้อยแล้วมา Past ลงในโปรแกรม Arduino
- สั่ง Save และทำการคอมไพล์
- เลือกคำสั่ง Upload to I/O Board ตัวซอฟต์แวร์ Arduino จะทำการอัพโหลดภาษาเครื่องที่ได้จากการคอมไพล์ลงสู่บอร์ดไมโครคอนโทรลเลอร์

7. ข้อมูลสำหรับการลงปฏิบัติการ

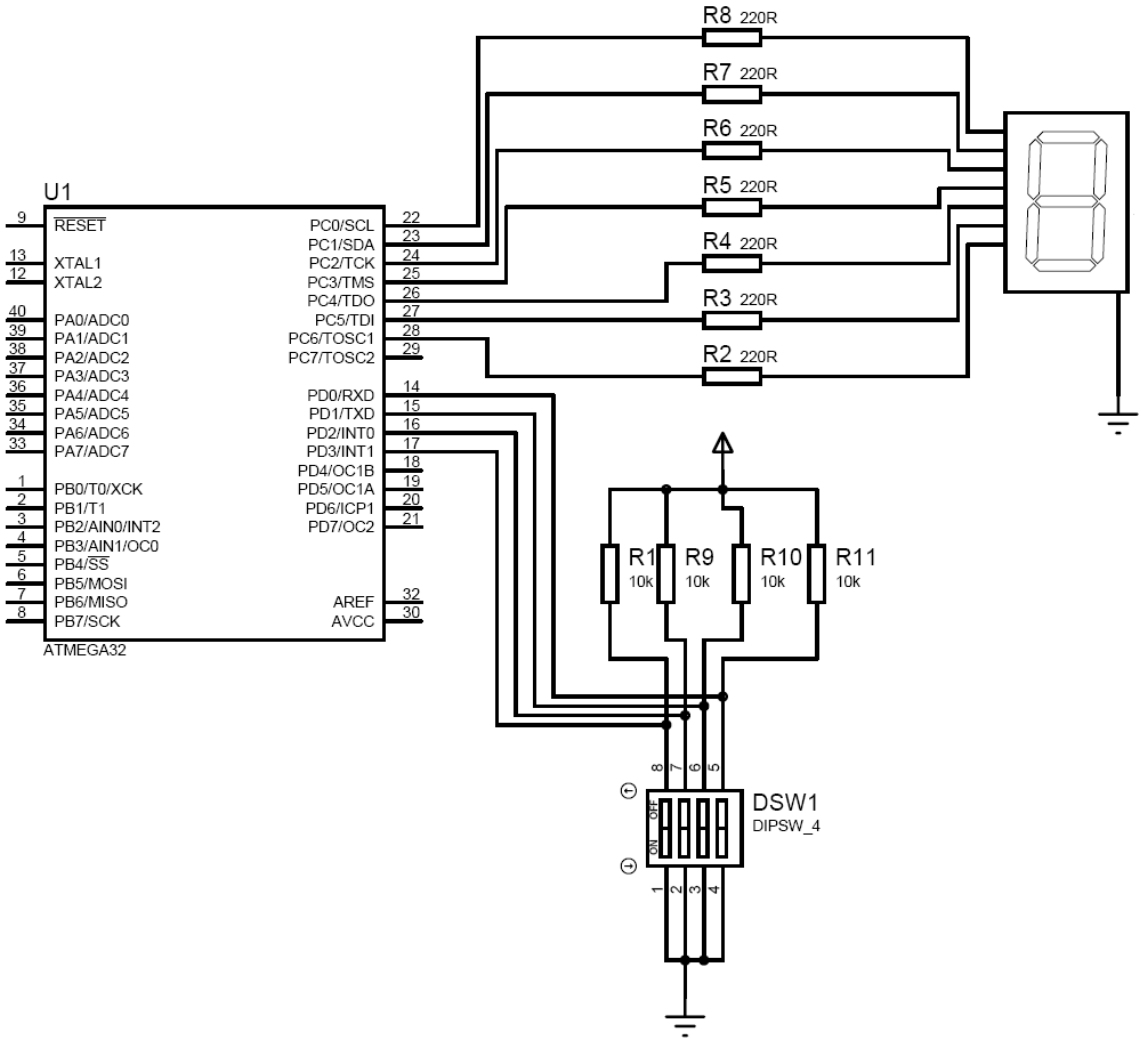
ให้นักศึกษาศึกษาข้อมูลการเขียนโปรแกรมภาษาแอสเซมบลี้และภาษาซี ตลอดจนข้อมูลรายละเอียดของสถาปัตยกรรม AVR จากเ1อกสารต่อไปนี้เพิ่มเติมก่อนที่จะมาทำการทดลองในห้องปฏิบัติการ ซึ่งสามารถดาวน์โหลดเอกสารได้จาก LMS

- Atmel Corporation, “AVR assembler user guide,”
  ดาวน์โหลดได้ที่http://lms.psu.ac.th/mod/resource/view.php?id=66518
- Atmel Corporation, AVR Datasheet: ATmega series,
  ดาวน์โหลดได้ที่ http://lms.psu.ac.th/mod/resource/view.php?id=66519
- Lam Phung, “Getting started with C Programming for the ATMEL AVR Microcontroller,”
  ดาวน์โหลดได้ที่ http://lms.psu.ac.th/mod/resource/view.php?id=72018

8. การทดลอง

8.1 ทดลองเขียนโปรแกรมภาษาแอสเซมบลี้เพื่ออ่านค่าจากสวิทช์แสดงผลทาง 7-segment LED

จากวงจรในรูปที่ 1.9 จะเห็นว่าซีพียู ATMEGA32 เชื่อมต่อกับแอลอีดีแบบ 7 เซกเมนต์ทางพอร์ต C และ เชื่อมต่อกับดิปสวิทช์จำนวน 4 ตัวกับสี่บิตล่างของพอร์ต B โปรแกรมในรูปที่ 1.10 เป็นโค้ดภาษาแอสเซมบลี้ซึ่งทำหน้าที่สั่งการให้ซีพียู AVR ทำการอ่านค่าจากดิปสวิทช์ จากนั้นทำการกรอง 4 บิตบนที่ไม่ใช้ทิ้งไปและทำการแปลงค่าไบนารีที่ได้ ซึ่งมีค่า 0-15 ไปแสดงผลทางแอลอีดี 7 เซกเมนต์ค่า 0-F ให้นักศึกษาเขียนโปรแกรมภาษาแอสเซมบลี้ดังกล่าวลงในซอฟต์แวร์ AVR Studio จากนั้นใช้คำสั่ง Build เพื่อแปลงภาษาแอสเซมบลี้ให้เป็นภาษาเครื่องของ AVR ซึ่งจะได้ไฟล์เอาต์พุตนามสกุล .HEX



รูปที่ 1.9 วงจรอ่านค่าสวิทช์ออกแสดงผลทางแอลอีดี 7 เซกเมนต์

```
.INCLUDE "m32def.inc"

.EQU    ALL_PIN_OUT = 0xFF
.EQU    ALL_PIN_IN  = 0x00
.DEF    VAR_A       = r16
.DEF    TMP         = r17

;-----------------------------------------------------------
;---code segment
;-----------------------------------------------------------
.CSEG
.ORG 0x0000
            ldi     VAR_A,ALL_PIN_OUT
            out     DDRC,VAR_A

            ldi     VAR_A,ALL_PIN_IN
            out     DDRD,VAR_A

            ldi     TMP, 0x00

MAIN:       ;----read 4 switches from PORT D
            in      VAR_A,PIND
            andi    VAR_A,0x0F

            ;---convert using look-up table
            ldi     ZL, low(TB_7SEGMENT*2)
            ldi     ZH, high(TB_7SEGMENT*2)

            add     ZL,VAR_A
            adc     ZH,TMP
            lpm

            out     PORTC,r0
            rjmp    MAIN
    ;----------------------------------
    ;---TABLE for 7-segment display
    ;----------------------------------
                    ;     hgfedcba    hgfedcba
    TB_7SEGMENT:    .DB 0b00111111, 0b00000110      ;0 and 1        --a--
                    .DB 0b01011011, 0b01001111      ;2 and 3        f    b
                    .DB 0b01100110, 0b01101101      ;4 and 5        --g--
                    .DB 0b01111101, 0b00000111      ;6 and 7        e    c
                    .DB 0b01111111, 0b01101111      ;8 and 9        --d--
                    .DB 0b01110111, 0b01111100      ;A and B
                    .DB 0b00111001, 0b01011110      ;C and D
                    .DB 0b01111001, 0b01110001      ;E and F

;-----------------------------------------------------------
;---data segment
;-----------------------------------------------------------
.DSEG
```

รูปที่ 1.10 โปรแกรมภาษาแอสเซมบลี้สำหรับอ่านค่าจากสวิทช์แสดงผลทาง 7-segment LED

8.2 ทดลองเขียนโปรแกรมภาษาซีเพื่ออ่านค่าจากสวิทช์แสดงผลทาง 7-segment LED

โปรแกรมในรูปที่ 1.11 เป็นโค้ดภาษาซีซึ่งทำหน้าที่สั่งการให้ซีพียู AVR ทำการอ่านค่าจากดิปสวิทช์ จากนั้นทำการกรอง 4 บิตบนที่ไม่ใช้ทิ้งไปและทำการแปลงค่าไบนารีที่ได้ ซึ่งมีค่า 0-15 ไปแสดงผลทางแอลอีดี 7 เซกเมนต์ค่า 0-F ซึ่งจะเห็นว่าโปรแกรมภาษาซีดังกล่าวมีหน้าที่การทำงานเหมือนกันกับโค้ดภาษาแอสเซม-บลี้ซึ่งแสดงในรูปที่ 1.10 ให้นักศึกษาเขียนโปรแกรมภาษาซีดังกล่าวลงในซอฟต์แวร์ AVR Studio จากนั้นใช้คำสั่ง Build เพื่อแปลงซอร์สโค้ดภาษาซีให้เป็นภาษาเครื่องของ AVR ซึ่งจะได้ไฟล์เอาต์พุตนามสกุล .HEX ต่อจากนั้นให้ทำการใช้โปรแกรม Arduino ทำการอัพโหลดโปรแกรมลงบนบอร์ด Diecimila ซึ่งเชื่อมต่อกับวงจรสวิทช์และแอลอีดี 7 เซกเมนต์ดังรูปที่ 1.8

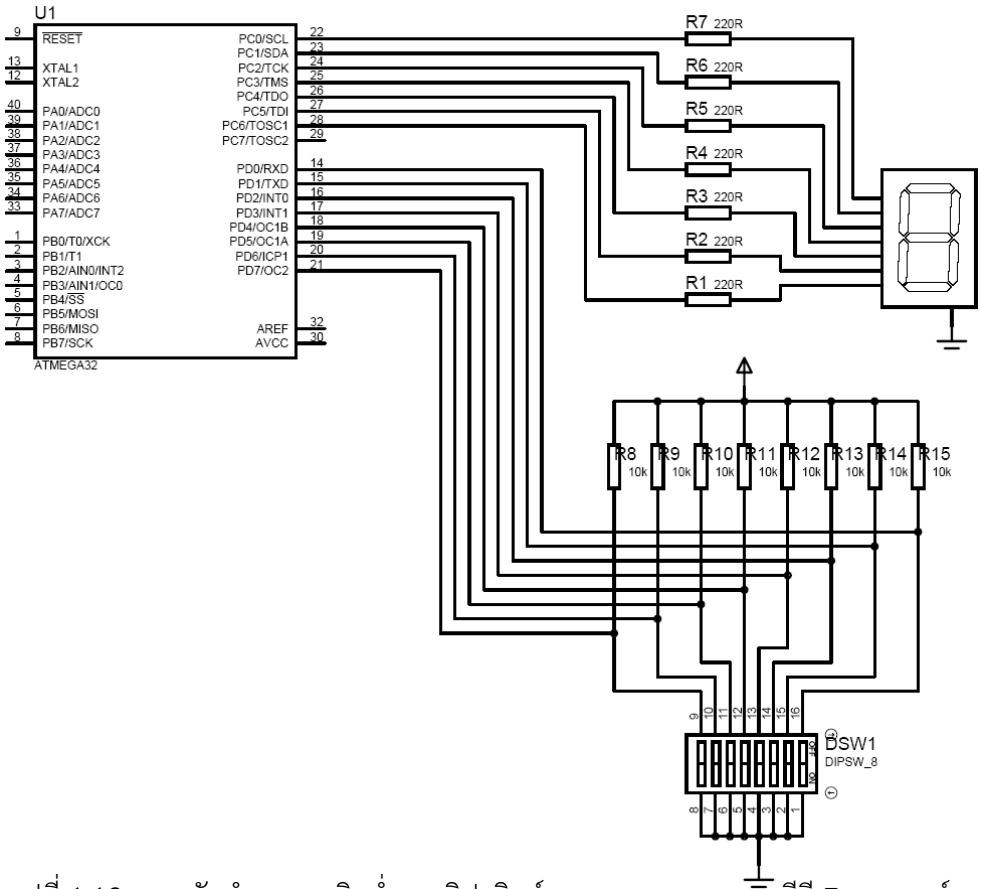
```
#include <avr/io.h>
int main(void)
{
    unsigned char LOOKUPTB[] = { 0b00111111,
                                 0b00000110,
                                 0b01011011,
                                 0b01001111,
                                 0b01100110,
                                 0b01101101,
                                 0b01111101,
                                 0b00000111,
                                 0b01111111,
                                 0b01101111,
                                 0b01110111,
                                 0b01111100,
                                 0b00111001,
                                 0b01011110,
                                 0b01111001,
                                 0b01110001 };

    unsigned char DISPLY;
    unsigned char SWITCH_V;

    DDRC = 0xFF;     //port C = output
    DDRD = 0x00;     //port D = input
    while(1)
    {
        SWITCH_V = PIND;
        SWITCH_V &= 0x0F;
        DISPLY = LOOKUPTB[SWITCH_V];
        PORTC = DISPLY;
    }
}
```

รูปที่ 1.11 โค้ดภาษาซีสำหรับอ่านค่าจากสวิทช์แสดงผลทาง 7-segment LED

Checkpoint 1	ลายเซ็นผู้คุมแลบ	วัน-เดือน-ปี
#checkpoint1.1 ขนาดของไฟล์นามสกุล .HEX ที่ได้จากต้นฉบับภาษาแอสเซมบลี้	.....	.....
#checkpoint1.2 ขนาดของไฟล์นามสกุล .HEX ที่ได้จากต้นฉบับภาษาซี	.....	.....
#checkpoint1.3 ต่อวงจร Dip-Switch และ 7-Segment LED เข้ากับบอร์ดไมโครคอนโทรลเลอร์ดังรูปที่ 1.9 พร้อมทั้งบันทึกโปรแกรมที่ได้จากการคอมไพล์โค้ดภาษาซีในรูป 1.11 ลงบนบอร์ด ทดลองปรับดิปสวิทช์ และสังเกตผลลัพธ์การประมวลผลที่ 7-Segment LED	.....	.....

8.3 ทดลองเขียนโปรแกรมภาษาซีเพื่อนับลอจิกต่ำจากดิปสวิทช์

จากวงจรในรูปที่ 1.12 จงเขียนโปรแกรมด้วยภาษาแอสเซมบลี้และภาษาซีเพื่อที่จะทำการนับจำนวนลอจิกต่ำซึ่งอ่านได้จากดิปสวิทช์ที่ต่ออยู่กับพอร์ต D ออกแสดงผลบนแอลอีดี 7 เซกเมนต์ซึ่งต่ออยู่กับพอร์ต C ให้ใช้โปรแกรม AVRStudio ทำการดีบักโปรแกรมจนแน่ใจว่าผลลัพธ์การทำงานถูกต้อง จากนั้นจึงนำโปรแกรมเวอร์ชั่นภาษาซีมาคอมไพล์ด้วยโปรแกรม Arduino 022 แล้วทำการอัพโหลดโปรแกรมลงบนบอร์ด Diecimila ซึ่งเชื่อมต่อกับวงจรสวิทช์และแอลอีดี 7 เซกเมนต์จริง



รูปที่ 1.12 วงจรนับจำนวนลอจิกต่ำจากดิปสวิทช์ออกแสดงผลทางแอลอีดี 7 เซกเมนต์

Checkpoint 2	ลายเซ็นผู้คุมแลบ	วัน-เดือน-ปี
#checkpoint2.1 แสดงให้เห็นว่านักศึกษาสามารถดีบักโปรแกรมภาษาแอสเซมบลี้ด้วยซอฟต์แวร์ AVRStudio ได้ ..... #checkpoint2.2 แสดงให้เห็นว่านักศึกษาสามารถดีบักโปรแกรมภาษาซีด้วยซอฟต์แวร์ AVRStudio ได้ ..... #checkpoint2.3 ต่อวงจร Dip-Switch และ 7-Segment LED เข้ากับบอร์ดไมโครคอนโทรลเลอร์ดังรูปที่ 1.12 พร้อมทั้งบันทึกโปรแกรมที่ได้จากการคอมไพล์โค้ดภาษาซีที่ได้ใน checkpoint 2.2 ลงบนบอร์ด ทดลองปรับดิปสวิทช์ และสังเกตผลลัพธ์การประมวลผลที่ 7-Segment LED	....	....

9. คำถามท้ายการทดลอง

- ให้นักศึกษาใช้คำสั่ง Disassembler ในโปรแกรม AVRStudio เพื่อดูภาษาแอสเซมบลี้ที่ได้จากการแปลงโปรแกรมภาษาซีในรูปที่ 1.11 แล้วเปรียบเทียบกับภาษาแอสเซมบลี้ที่เขียนขึ้นในรูปที่ 1.10 จงวิเคราะห์ความแตกต่างของโค้ดภาษาแอสเซมบลี้ทั้งสอง และสรุปเปรียบเทียบข้อดีข้อเสียของการเขียนโปรแกรมด้วยภาษาแอสเซมบลี้และภาษาซี
- จงเขียนออกแบบวงจรพร้อมทั้งเขียนโปรแกรมภาษาแอสเซมบลี้และภาษาซีของ AVR เพื่อทำการอ่านค่าจากดิปสวิทช์ 4 ตัว และแสดงผลค่าตัวเลขที่อ่านจากสวิทช์ออกสู่แอลอีดี 7 เซกเมนต์ 2 หลัก โดยกำหนดให้ค่า 4 บิตที่อ่านจากสวิทช์เป็นตัวเลขจำนวนเต็มแบบมีเครื่องหมาย การทำงานของโปรแกรมจะแสดงผลตามตารางที่ 1.1

ตารางที่ 1.1 การแสดงผลของแอลอีดีเมื่อมีอินพุตสภาวะต่างๆ

อินพุตที่อ่านจากสวิทช์	ค่าที่แสดงผลบนแอลอีดี 7 เซกเมนต์จำนวน 2 หลัก
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

10. เอกสารอ้างอิง

- ปัญญยศ ไชยกาฬ, 2553, เอกสารประกอบการสอนรายวิชา 241-210 สถาปัตยกรรมไมโครโพรเซสเซอร์และภาษาแอสแซมบลี. http://lms.psu.ac.th/course/view.php?id=2999
- Steven F. Barrett, Daniel J. Pack, “Atmel AVR microcontroller primer: programming and interfacing,” Morgan and Claypool, 2008.
- Richard H. Barnett, Larry O'Cull, Sarah Cox, “Embedded C programming and the Atmel AVR,” Thomson Delmar Learning, 2006.
