

Lab 1 : การใช้งานภาษาแอสเซมบลีกับชิพ AVR

1.1 แนะนำชิพ AVR

ไมโครคอนโทรลเลอร์ เป็นหน่วยประมวลผลขนาดเล็ก ซึ่งรวมเอาความสามารถของไมโครโปรเซสเซอร์กับอุปกรณ์เชื่อมต่อภายนอก เช่น หน่วยความจำ อุปกรณ์อินพุตเอาต์พุต วงจรกำเนิดสัญญาณนาฬิกา เข้ามารวมอยู่ในไอซีชิพเพียงตัวเดียว ส่งผลให้ระบบควบคุมซึ่งใช้ไมโครคอนโทรลเลอร์เป็นหน่วยประมวลผลมีขนาดของวงจรควบคุมเล็กและสะดวกต่อการใช้งาน

แม้ว่าระบบไมโครคอนโทรลเลอร์จะมีหลายๆ อุปกรณ์อยู่ในไอซีชิพตัวเดียว แต่ก็มีความสามารถจำกัด เมื่อเทียบกับระบบซึ่งใช้ไมโครโปรเซสเซอร์ ยกตัวอย่างเช่น หน่วยความจำ จะมีให้ใช้เพียงไม่มากนัก ดังนั้นระบบไมโครคอนโทรลเลอร์จะเหมาะสมสำหรับการควบคุมงานที่ไม่ซับซ้อนและต้องการความสามารถในการประมวลผลที่ไม่สูงมากนัก ยกตัวอย่างเช่นในระบบ Embedded system หรือระบบสมองกลฝังตัว เช่น เครื่องซักผ้า เตอบไมโครเวฟ เครื่องปรับอากาศ เป็นต้น

สถาปัตยกรรมของไมโครคอนโทรลเลอร์ที่ใช้งานอยู่ในปัจจุบันมีให้เลือกใช้งานมากมายหลายตัว ยกตัวอย่างเช่น

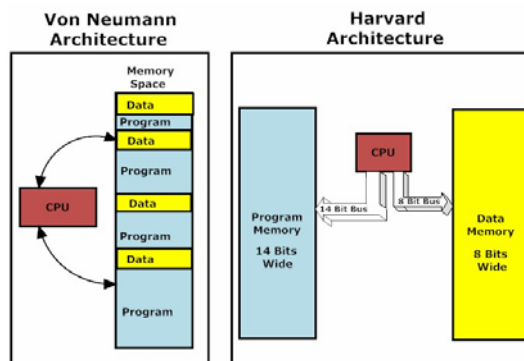
- MCS-51 (8-bit) ออกแบบโดย intel
- MCS-96 (16-bit) ออกแบบโดย intel
- PIC ออกแบบโดย Microchip Technology
- AVR ออกแบบโดย Atmel
- ARM ออกแบบโดย ARM Holdings
- 68HC11 ออกแบบโดย Motorola
- Rabbit 2000 ออกแบบโดย Rabbit Semiconductor

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

สถาปัตยกรรมของ AVR แบ่งออกเป็น 2 ประเภทคือ

- 8-bit AVR
- 32-bit AVR

ในแลบนี้จะกล่าวถึงสถาปัตยกรรมของ AVR ขนาด 8 บิตเท่านั้น โดยในสถาปัตยกรรม AVR ออกแบบโดย ATMEAL เมื่อปี 1996 เป็นชิพแบบ RISC (Reduced Instruction Set Computer) มีสถาปัตยกรรมการต่อหน่วยความจำแบบ Harvard ซึ่งแยกหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลออกจากกันโดยเด็ดขาด ดังแสดงในรูปที่ 1.1 โดยใช้หน่วยความจำแบบ Flash สำหรับเป็นหน่วยความจำโปรแกรม และใช้หน่วยความจำแบบ SRAM สำหรับหน่วยความจำข้อมูล และนอกจากนี้ยังมีหน่วยความจำแบบ EEPROM ซึ่งสามารถเก็บข้อมูลเอาไว้ได้โดยไม่ต้องมีไฟเลี้ยงอีกด้วย



รูปที่ 1.1 เปรียบเทียบการจัดการหน่วยความจำของสถาปัตยกรรมแบบ Von-Neumann และ Harvard

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

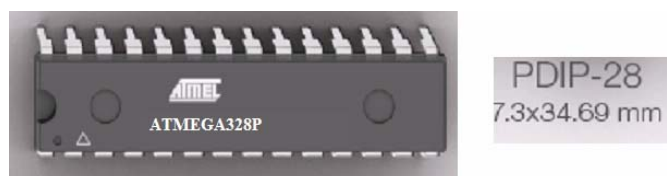
ลักษณะเด่นของสถาปัตยกรรม AVR คือ คำสั่งส่วนใหญ่สามารถทำงานได้เสร็จภายใน 1 clock cycle ตัวชิพ AVR ขนาด 8 บิตจะแบ่งออกเป็นประเภทการใช้งานได้ 5 กลุ่ม ได้แก่

- tinyAVR เป็นชิพในรุ่นเล็ก ซึ่งต้องการความลึกเกทที่ต่ำของวงจร โดยเหมาะกับระบบควบคุมขนาดเล็กๆ ที่ต้องการหน่วยความจำและวงจรสนับสนุนไม่มากนัก ชิปในรุ่นนี้จะมีราคาถูกกว่ากลุ่มอื่น
- megaAVR จะมีชื่ออีกอย่างว่า ATmega โดยมีวงจรสนับสนุนภายในเพิ่มเติมตลอดจนเพิ่มขนาดของหน่วยความจำให้ใช้งานมากกว่าตระกูล Tiny เหมาะกับงานควบคุมทั่วๆ ไป

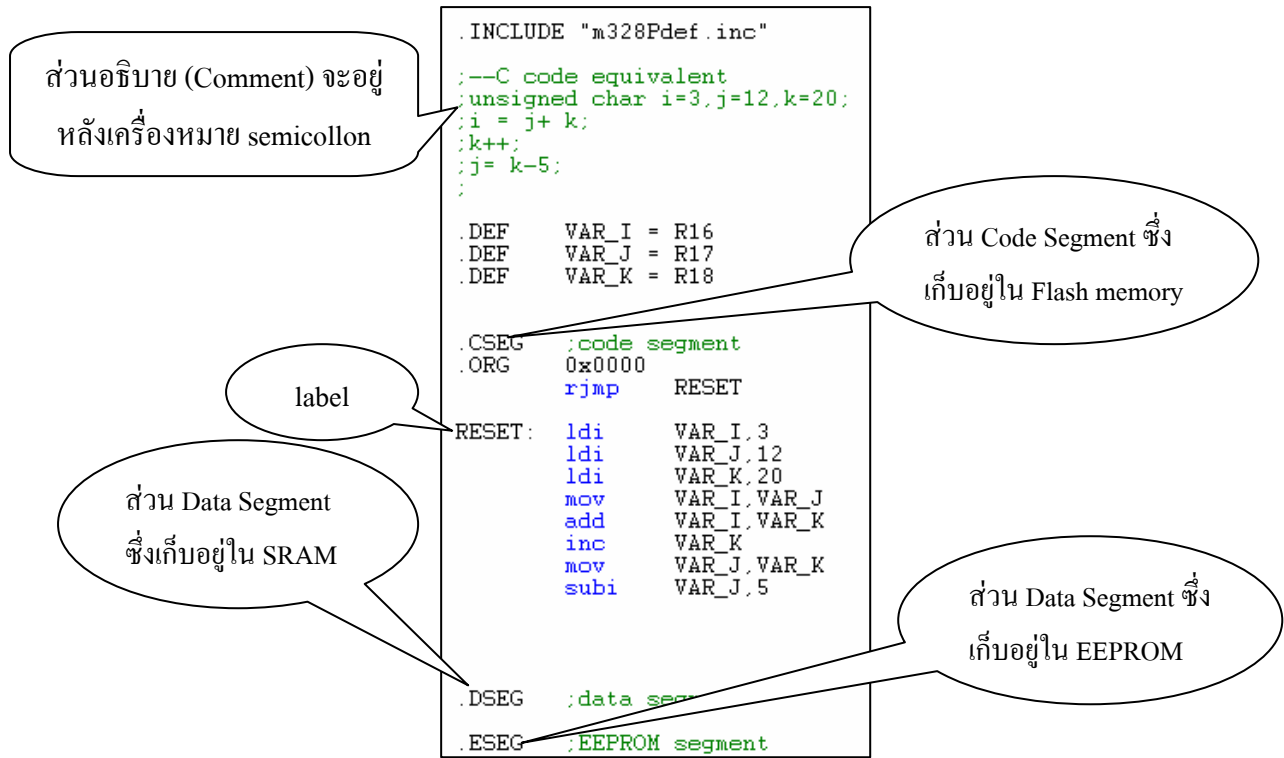
- XMEGA เพิ่มความละเอียดของวงจร A/D จากปกติมีความละเอียด 10 บิตในรุ่นเล็กกว่าเป็น 12 บิต และวงจร DMA controller ซึ่งช่วยลดภาระของซีพียูในการควบคุมการรับส่งข้อมูลระหว่างอุปกรณ์ I/O กับหน่วยความจำ
- FPSLIC (AVR core with FPGA) สำหรับงานที่ต้องการควบคุมที่ต้องการความยืดหยุ่นในขั้นตอนการออกแบบและพัฒนา โดยผู้ออกแบบสามารถออกแบบวงจรในระดับฮาร์ดแวร์เพิ่มเติมด้วยภาษาบรรยายฮาร์ดแวร์ (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 ในหัวข้อเลข 3 เทอม 2 นี้ จะเลือกใช้ซีพียูรุ่น ATmega328P ให้นักศึกษาใช้เป็นหลัก ซึ่งมีคุณสมบัติดังนี้

- หน่วยความจำโปรแกรมแบบ FLASH ขนาด 32 กิโลไบต์
- หน่วยความจำข้อมูลแบบ SRAM ขนาด 2 กิโลไบต์
- หน่วยความจำข้อมูลแบบ EEPROM ขนาด 1 กิโลไบต์
- สนับสนุนการเชื่อมต่อแบบ I²C bus
- พอร์ตอินพุตเอาต์พุตจำนวน 23 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
- ทำงานได้ตั้งแต่ย่านแรงดัน 1.8-5.5 Volts
- ความถี่ใช้งานสูงสุด 20 MHz



รูปที่ 1.2 ซีพียู ATMEGA328P แบบตัวถัง PDIP ขนาด 28 ขา



รูปที่ 1.3 โครงสร้างภาษาแอสเซมบลีของสถาปัตยกรรม AVR

1.2 แนะนำภาษาแอสเซมบลีของ AVR

โครงสร้างภาษาแอสเซมบลีของ AVR แสดงให้เห็นดังรูปที่ 1.1 ซึ่งจะเห็นว่ามีความแตกต่างจากภาษาแอสเซมบลีของสถาปัตยกรรมอื่นๆ ไม่มากนัก ในการเขียนโปรแกรมภาษาแอสเซมบลี เราจะต้องทำการใช้ชุดคำสั่งของซีพียูในการเข้าถึงหน่วยความจำและข้อมูลในรีจิสเตอร์โดยตรง ส่งผลให้ภาษาแอสเซมบลีมีความยุ่งยากในการใช้งานมากกว่าระดับสูงทั่วไป อย่างไรก็ตามก็ถือว่าภาษาแอสเซมบลีมีข้อดีที่ภาษาอื่นตรงที่ขนาดของโปรแกรมนั้นมีขนาดเล็กมาก และมีความเร็วในการทำงานที่สูงกว่าภาษาอื่นๆ การเรียนรู้ภาษาแอสเซมบลีช่วยให้นักศึกษาสามารถเข้าใจการทำงานของไมโครโปรเซสเซอร์ได้เป็นอย่างดี การเข้าใจภาษาแอสเซมบลีจะช่วยให้นักศึกษาสามารถที่จะดีบั๊ก (Debug) เพื่อทำการตรวจสอบการทำงานของโปรแกรมในกรณีที่โปรแกรมที่เขียนขึ้นด้วยภาษาระดับสูงมีปัญหาได้ สำหรับรายละเอียดเชิงลึกในการเขียนโปรแกรมภาษาแอสเซมบลี ขอให้นักศึกษาได้ศึกษาจากเอกสาร AVR Assembler User Guide ในเอกสารเล่มนี้จะทำการยกตัวอย่างการเขียนภาษาแอสเซมบลีอย่างง่ายเพื่อเป็นพื้นฐานในการนำซีพียูไปประยุกต์ใช้งานระดับสูงขึ้นไปในโอกาสต่อไป

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

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)

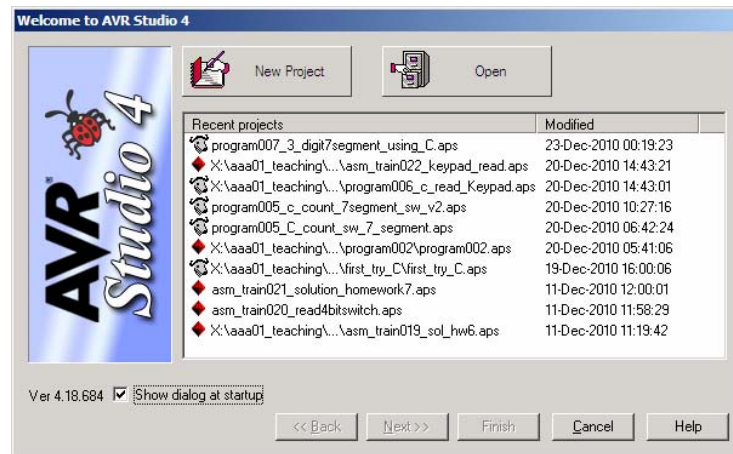
1.3 แนะนำซอฟต์แวร์ AVRStudio

โปรแกรม AVRStudio เป็นซอฟต์แวร์ที่พัฒนาโดย ATMEL ซึ่งแจกจ่ายให้ใช้งานได้ฟรี ซึ่งใช้เป็นสภาพแวดล้อมในการพัฒนาโปรแกรมด้วยภาษาแอสเซมบลีหรือภาษาซีก็ได้ ในฉบับนี้จะยกตัวอย่างการใช้ซอฟต์แวร์ AVRStudio ในการพัฒนาโปรแกรมภาษาแอสเซมบลี

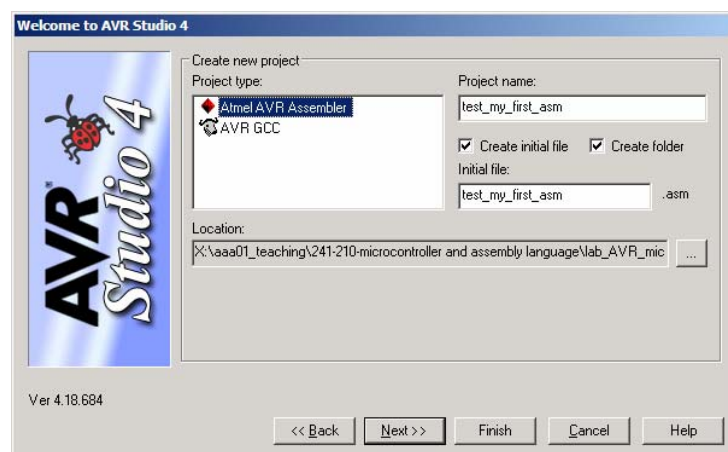
เรียกใช้งานโปรแกรมจาก Start menu->All Programs->ATmel AVR Tools->AVR Studio 4



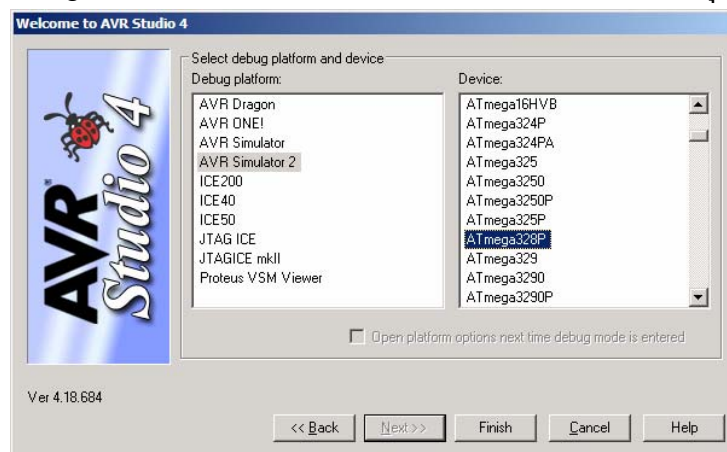
โปรแกรมจะแสดงหน้าต่าง ดังรูป ให้เลือกกดปุ่ม New Project



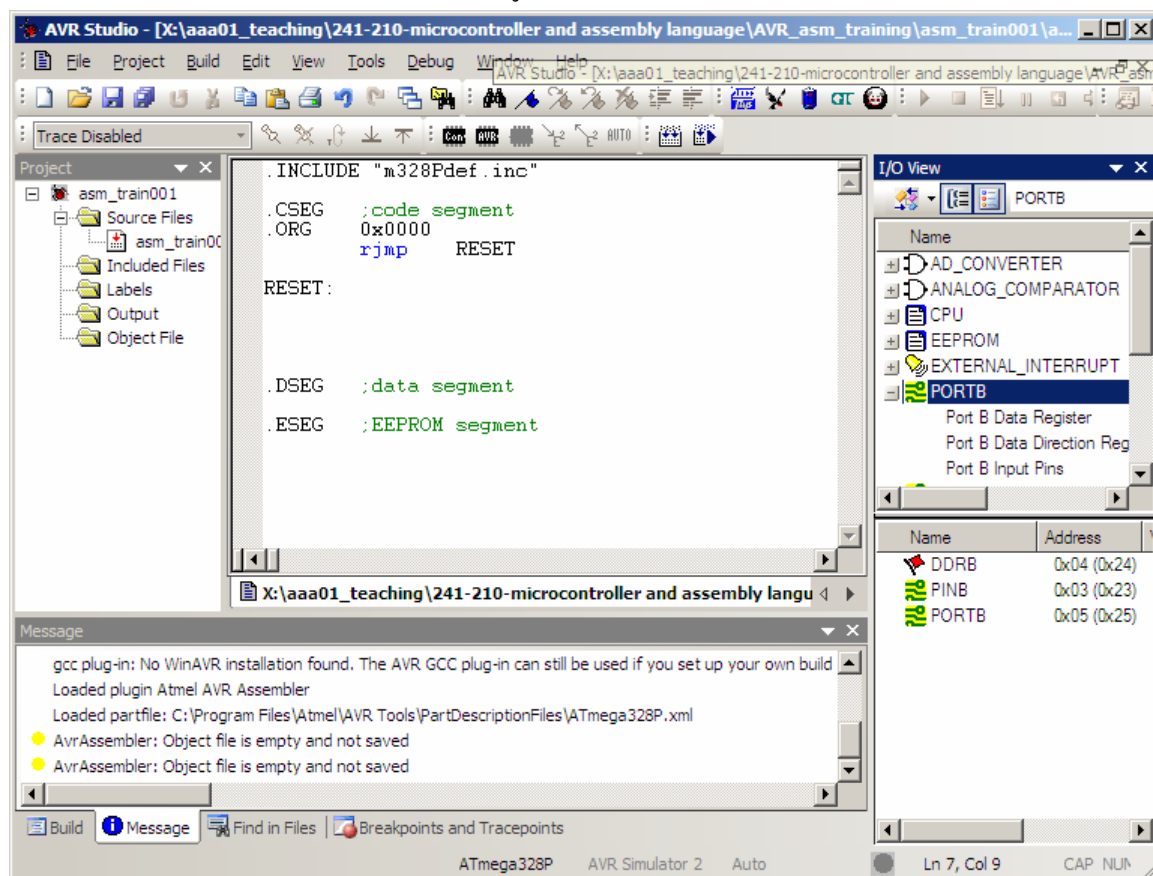
เลือกสร้างโปรเจกต์ด้วยภาษาแอสเซมบลี ดังรูป โดยป้อนชื่อโปรเจกต์ลงไป



เลือกซีพียูรุ่น ATmega328P และสภาพแวดล้อม AVR Simulator2 จากนั้นกดปุ่ม Finish

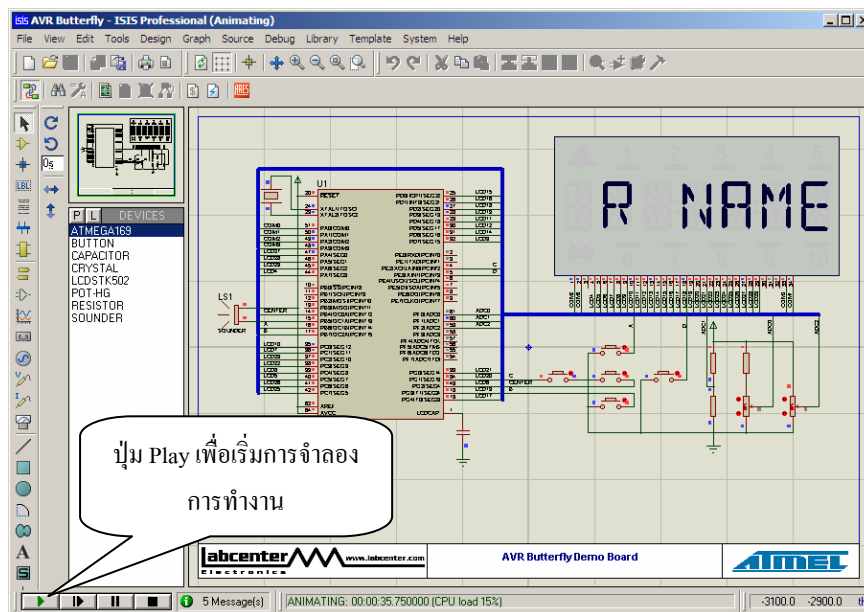


จะขึ้นสภาพแวดล้อมการพัฒนาโปรแกรมดังรูป



1.4 แนะนำซอฟต์แวร์ชุด Proteus


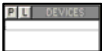
เป็นซอฟต์แวร์สำหรับจำลองการทำงานของวงจรไฟฟ้า ซึ่งสามารถจำลองการทำงานของวงจรได้ตั้งแต่วงจรอนาล็อก วงจรดิจิทัลตลอดจนถึงไมโครโปรเซสเซอร์และไมโครคอนโทรลเลอร์ โดยผู้ที่ทดลองสามารถนำไฟล์นามสกุล .HEX ซึ่งได้จากการแอสเซมบลีโปรแกรมที่เขียนขึ้นด้วยภาษาแอสเซมบลีมาทำการจำลองบนสภาพแวดล้อมการเชื่อมต่อไมโครคอนโทรลเลอร์กับอุปกรณ์ภายนอกเสมือนจริงได้ ตัวชุดโปรแกรม Proteus ประกอบไปด้วยซอฟต์แวร์ย่อยหลายตัว ในเล่มนี้จะใช้โปรแกรมย่อยชื่อ ISIS ในการจำลองการทำงานของไมโครคอนโทรลเลอร์ AVR



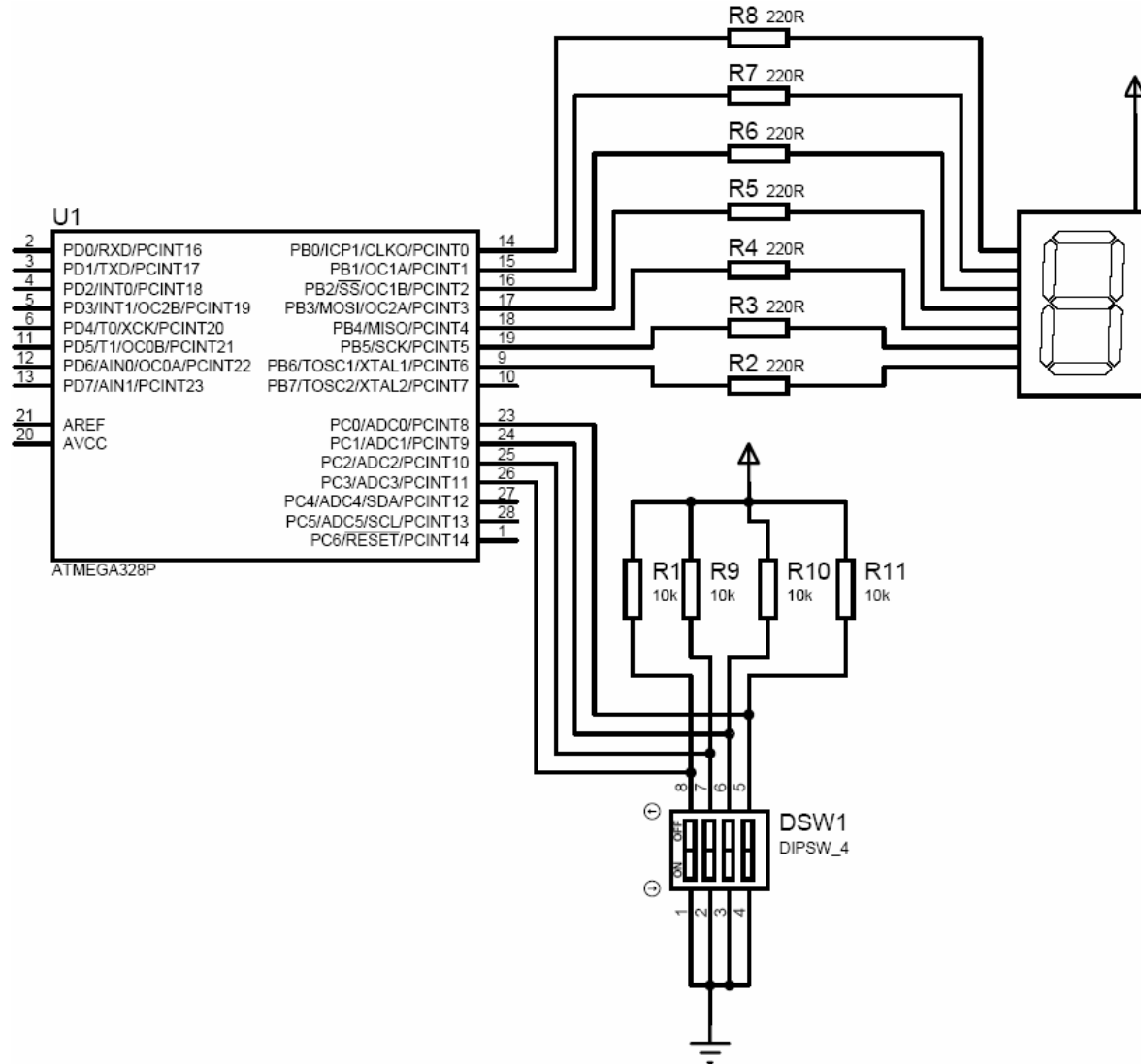
รูปที่ 1.6 โปรแกรม ISIS 7.0 สำหรับจำลองการทำงานของไมโครคอนโทรลเลอร์

1.5 การทดลอง

1.5.1 การออกแบบวงจรอ่านค่าจากสวิทช์และแสดงผลออกทาง 7-segment LED

ให้นักศึกษาเปิดโปรแกรม ISIS คลิกเลือก New Design เลือกขนาดของกระดาษเป็น Landscape A4 ทำการวาดวงจรเพื่อทำการอ่านค่าจากสวิทช์ดังรูปที่ 1.7 กดปุ่ม  จากนั้นดับเบิลคลิกที่  เพื่อเลือกใช้อุปกรณ์จากไลบรารีต่อไปนี้

Category	Sub-Category	Device	คำอธิบาย
Microprocessor ICs	AVR Family	ATMEGA328P	ไมโครคอนโทรลเลอร์ AVR
Optoelectronics	7-Segment Displays	7-SEG-COM-CAT	ชนิด Common Cathode
Switches & Relays	Switches	DIPSW-4	Dip Switch 4 ตัวใน 1 package
Resistors	Generic	RES	ตัวต้านทาน



รูปที่ 1.7 วงจรอ่านค่าจากสวิตช์แสดงผลทาง 7-segment LED

รายการ	ลายเซ็น	วัน-เดือน-ปี
#checkpoint1 วาดวงจรดังรูปที่ 1.7 โดยใช้โปรแกรม ISIS7.0		

1.5.2 การเขียนโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าจากสวิตช์แสดงผลทาง 7-segment LED

เขียนโปรแกรมภาษาแอสเซมบลีเพื่อใช้ควบคุมไมโครคอนโทรลเลอร์ของวงจรในรูปที่ 1.7 โดยตัวโปรแกรมแสดงให้เห็นในรูปที่ 1.8 การทำงานของตัวโปรแกรม จะทำการอ่านค่าจากคิปสวิตช์ขนาด 4 บิต เข้ามาทางพอร์ต C บิตที่ 0-3 และทำการแปลงค่าไบนารีที่ได้ ซึ่งมีค่า 0-15 ไปแสดงผลทางแอลอีดี 7 เซกเมนต์ ค่า 0-F ให้นักศึกษาเขียนโปรแกรมภาษาแอสเซมบลีด้วย AVR Studio จากนั้นใช้คำสั่ง Build เพื่อแปลงภาษาแอสเซมบลีให้เป็นภาษาเครื่องของ AVR ซึ่งจะได้ไฟล์เอาต์พุตนามสกุล .HEX

```
.INCLUDE "m328Pdef.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    DDRB,VAR_A           ;set port B as output

        ldi    VAR_A,ALL_PIN_IN
        out    DDRC,VAR_A           ;set PORT C as input

        ldi    TMP,0x00              ;TMP <= 0

MAIN:    ;---read 4 switches from portC(lower4bits) and keep in VAR_A
        in     VAR_A,PINC             ;4 switches are connected to bit 0-3
        andi   VAR_A,0x0F             ;upper 4 bits are ignored

        ;---look the table up
        ldi    ZL,low(TB_7SEGMENT*2) ;load Z register low
        ldi    ZH,high(TB_7SEGMENT*2) ;load Z register high


        add    ZL,VAR_A
        adc    ZH,TMP                 ; Z <= Z + VAR_A
        lpm                                ;R0 <- [Z]

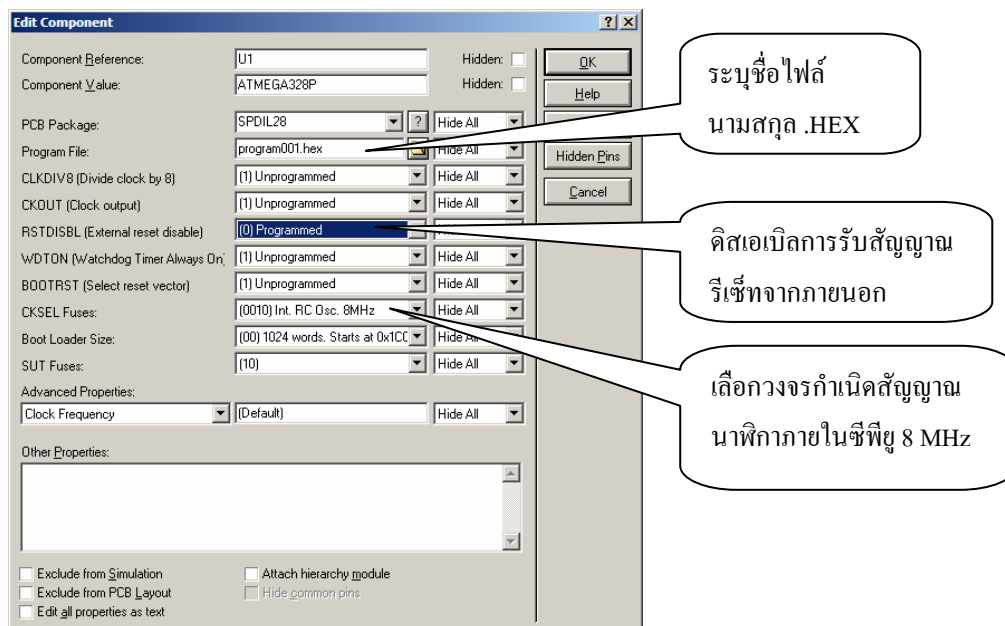
        com    r0
        out    PORTB,r0
        rjmp   MAIN

;-----
; Table for 7-segment display
;-----
TB_7SEGMENT:
        ;      hgfedcba  hgfedcba
        .DB 0b00111111, 0b000000110 ;0 and 1      --a--
        .DB 0b01011011, 0b01001111  ;2 and 3      f      b
        .DB 0b01100110, 0b01101101   ;4 and 5      --g--
        .DB 0b01111101, 0b000000111   ;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
;-----
; EEPROM segment
;-----
.ESEG
```

รูปที่ 1.8 โปรแกรมภาษาแอสเซมบลีสำหรับอ่านค่าจากสวิตช์แสดงผลทาง 7-segment LED

ในการใช้ซอฟต์แวร์ ISIS ทำการจำลองการทำงานของโปรแกรมภาษาแอสเซมบลีที่เขียนขึ้นนั้น สามารถทำได้โดยคลิกขวาที่ตัวไมโครคอนโทรลเลอร์ เลือก popup menu ชื่อ Edit Properties จะขึ้นไดอะล็อกบ็อกซ์ดังรูปที่ 1.9 ซึ่งผู้ทดลองต้องระบุชื่อไฟล์นามสกุล .HEX ที่ไมโครคอนโทรลเลอร์จะต้องโหลดขึ้นมาจำลองการทำงาน และให้สั่งทำการดิสเอเบิลการรีเซ็ตซีพียูจากภายนอกด้วย จากนั้นกดปุ่ม  เพื่อเริ่มจำลองการทำงานของซีพียู



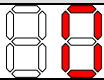
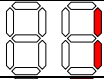
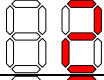
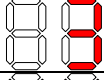
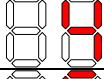

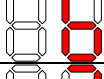

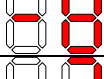
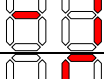
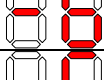

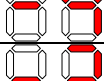
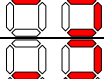

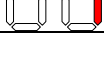
รูปที่ 1.8 การตั้งค่าควบคุมไมโครคอนโทรลเลอร์ในโปรแกรม ISIS

รายการ	ลายเซ็น	วัน-เดือน-ปี
#checkpoint2 ทดสอบจำลองการทำงานของโปรแกรมภาษาแอสเซมบลีกับโปรแกรม ISIS7.0		

1.5.3 โจทย์ปัญหาการออกแบบวงจรและเขียนโปรแกรมภาษาแอสเซมบลี

จงออกแบบวงจรไมโครคอนโทรลเลอร์ AVR สำหรับอ่านค่าจากคิปสวิทช์ 4 ตัว และแสดงผลค่าตัวเลขที่อ่านจากคิปสวิทช์ออกสู่แอลอีดี 7 เซกเมนต์ 2 หลัก โดยกำหนดให้ค่า 4 บิตที่อ่านจากคิปสวิทช์เป็นตัวเลขจำนวนเต็มแบบมีเครื่องหมาย โดยการทำงานของโปรแกรมจะแสดงผลตามตารางที่ 1.1

ตารางที่ 1.1 การแสดงผลของแอลอีดีเมื่อมีอินพุตสถานะต่างๆ

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

รายการ	ลายเซ็น	วัน-เดือน-ปี
#checkpoint3 วงจรที่ออกแบบโดย ISIS7.0		
#checkpoint4 ทดสอบโปรแกรมภาษาแอสเซมบลีกับวงจรที่ออกแบบด้วย โปรแกรม ISIS7.0		
