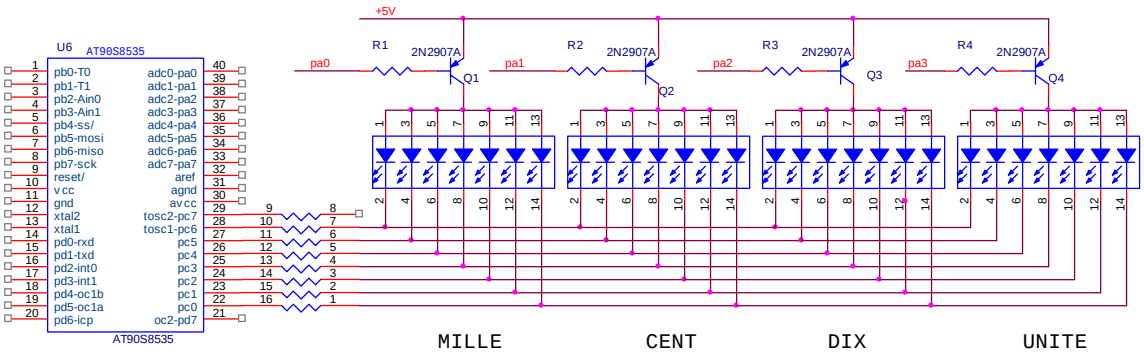
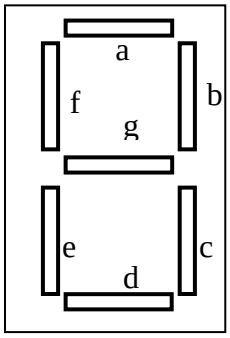
Examen module 3, année 05-06
Partie microcontrôleur AT90S8535 durée : 1h30 documents non autorisés

1) Dans la mise en œuvre d'un microcontrôleur Atmel AT90S8535 justifier, à partir d'un exemple judicieusement choisi, la raison pour laquelle on doit au moins sauvegarder le registre d'état (SREG), lors de l'exécution d'un sous-programme d'interruption. Vous donnerez un extrait de programme assembleur qui effectue cette opération (sauvegarde + restitution).

2) On désire réaliser un programme qui gère quatre afficheurs 7 segments à anodes communes (milliers, centaines, dizaines et unités) en mode multiplexé afin d'économiser des broches d'entrées-sorties du microcontrôleur. Le nombre à afficher va de 0000 à 9999. Le câblage des afficheurs est représenté ci-dessous.





a relié à pc0, b => pc1, c => pc2, d => pc3, e => pc4, f => pc5, g => pc6
pa0 => afficheur des milliers, pa1 => afficheur des centaines
pa2 => afficheur des dizaines, pa3 => afficheur des unités

<u>exemple :</u> pour afficher 0 sur les milliers il faut :
pc0 = pc1 = pc2 = pc3 = pc4 = pc5 = 0 pc6 = 1
et pa0 = 0 (Q1 fermé), pa1 = pa2 = pa3 = 1 (Q2, Q3, Q4 ouverts)

Pour cela on décide de créer, à partir d'un timer, une horloge temps réel qui provoque une interruption toutes les 5 ms et au cours de laquelle on vient activer les afficheurs à tour de rôle.

2.1) Comment doivent-être initialisés les ports A et C ?
2.2) Proposer un algorigramme détaillé du sous-programme d'interruption qui gère l'affichage multiplexé. (On prendra 4 registres de travail M, C , D, U et on considérera que le codage 7 segments est déjà réalisé).
2.3) Ecrire le sous-programme correspondant en langage assembleur AT90S8535.

3) <u>Transcodage binaire => 7 segments</u>

On suppose maintenant que les quatre registres M, C, D, U contiennent les codes binaires des chiffres à afficher. On se propose de réaliser un sous-programme de conversion appelé « transcode » qui convertit un chiffre binaire dans le code 7 segments correspondant. (on chargera un registre **temp** avec la valeur binaire à convertir et le sous-programme retournera le résultat dans le même registre).

3.1) Compléter le tableau ci-dessous :

Chiffre décimal à afficher	Code binaire	Code 7 segments
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

3.2) Proposer un algorigramme détaillé du sous-programme qui effectue la conversion binaire => 7 segments.
3.3) Ecrire le sous-programme correspondant en langage assembleur AT90S8535.

Instructions disponibles

Arithmetic and Logic Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	Rd,Rr	Add with Carry	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
ADIW	Rd, K	Add Immediate To Word	Rd+1:Rd,K	Z,C,N,V,S	2
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Branch Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2

IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
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,C,N,V,H, S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H, S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H, S	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 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(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Data Transfer Instructions

Mnemonic	**Operands**	**Description**	**Operation**	**Flags**	**Cycles**
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*

LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	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-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2
ST	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-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3
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
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Bit and Bit-test Instructions

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
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

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Z=R31/R30 Y=R29/R28 X=R27/R26

Barême : Partie 1 = 2 points ; Partie 2 = 9 points ; Partie 3 = 8 points ; Présentation = 1 point

Nota : Une partie des instructions PUSH, POP, MOV, SBIC, SBIS, SBI, CBI, RJMP, OUT, IN, RETI, RET, CPI, BREQ, BRNE, LDI , CLR, LD, ADD suffisent pour répondre aux problèmes.