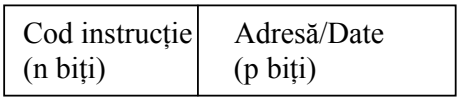
Tema proiect:

Realizarea unui simulator de AP 101. Functionarea este descrisa mai jos. Simulatorul va fi realizat in orice mediu de programare care permite aceasta.

Programarea automatelor programabile cu prelucrare la nivel de bit

Limbajul utilizat pentru programarea automatelor programabile este un limbaj de asamblare în cadrul căruia fiecărei instrucțiuni i se asociază o mnemonică unică. Programatorul scrie programul utilizând această mnemonică şi operanzii sau adresele asociate.

Programul sursă astfel obținut este convertit în cod maşină cu ajutorul programului de conversie. În cadrul acestei conversii fiecare instrucțiune din limbaj de asamblare din programul sursă este convertită în una sau mai multe instrucțiuni cod maşină. Formatul general al unei instrucțiuni în cod maşină este de forma generală din figură şi conține două câmpuri:



- Câmp rezervat pentru codul instrucțiunii
- Câmp rezervat pentru:

a. valoarea decodificată a adresei operandului curent, care poate fi de intrare sau de ieşire
b. valoarea decodificată a locației din memoria programului la care se execută saltul
c. o dată imediată de tipul valorii temporizării, în cazul când stabilirea temporizării se face software.

Câmpul rezervat pentru codul instrucțiunii conține referiri la modul de adresare specific instrucțiunii curente, adică la modul în care o adresă oarecare trebuie să fie interpretată.

La automatele programabile cu prelucrare la nivel de bit sunt definite două moduri de adresare:

1. Adresarea directă – prin care adresa operandului este specificată direct în instrucțiune. Adresarea directă nu necesită instrucțiuni multicuvânt intrucât întreg domeniul de adresabilitate este acoperit prin cei p biți ai câmpului de adresă ai unei instrucțiuni ce mai are n biți rezervați pentrul codul instrucțiunii.
2. Adresarea indexată – prin care adresa specificată în instrucțiune este relativa la un registru index. Acest tip de adresare face posibilă operarea automatelor programabile cu prelucrare la nivel de bit în modurile multiprogram fixe sau reconfigurabile.

Unitatea centrală a unui automat programabil cu prelucrare la nivel de bit este sincronizată de un semnal de ceas provenit de la un oscilator intern sau extern. Execuția unei instrucțiuni este definită de un ciclu de instrucțiune care conține uzual un singur ciclu maşină. Pe durata unui ciclu maşină sunt executate următoarele subcicluri:

- Subciclul de extragere în care unitatea centrală furnizează adresa unei instrucții din memoria program MP pe baza informației conținută în numărătorul de adresă NA. Adresa este decodificată şi instrucțiunea este extrasă din MP şi depusă în registrul instrucțiunii curente RIC.
- Subciclul de execuție în care instrucțiunea este decodificată şi operațiunea cerută este executată.

Testarea condițiilor şi transferul datelor

Instrucțiunile de testare detectează schimbările de stare ale semnalelor de intrare, de ieşire sau memorate temporar şi preiau aceste schimbări prin memorarea valorii testate în registrul acumulator al unității centrale. Semnalul testat poate fi transmis pe magistrala interna a automatului programabil în formă normală sau complementată.

Vom analiza operațiile pe care le efectuează un automat programabil cu prelucrare la nivel de bit, pornind de la setul de instrucțiuni al automatului programabil AP101. Pentru aceasta se vor utiliza următoarele simboluri:

1. adr p (R, E, I sau M)

2. (adr p)
3. ⟵ sensul de transfer
4. ⟷ schimbul între
5. - sau / complement

1. Este folosit pentru specificarea unei adrese exprimată pe p biți, ce poate fi adresa unui canal de intrare, de ieşire, a unui modul de temporizare/contorizare sau a unei locații din memoria RAM.

2. Specifică conținutul locației de memorie sau a canalului de intrare, ieşire sau temporizare cu adresa identică cu cea impusă între paranteze.

Ținând cont de acest mod de notare, vom avea:

- Instrucțiuni de testare a condiției, cu operațiile pe care le introduce

1. LD adr p (A) ⟵ (adr p)
2. LDC adr p (A) ⟵ $\overline{\text{(adr p)}}$

- încarcă în acumulator conținutul locației de adresă sau complementul locației de adresă respectivă.

- Instrucțiuni de transfer de date – permit salvarea unui rezultat curent aflat în acumulator într-o locație a memoriei RAM sau încărcarea unuia din bistabilele din structura canalelor de ieşire sau temporizare/contorizare cu conținutul acumulatorului.

3. STO adr p (adr p) ⟵ (A)
4. STOC adr p (adr p) ⟵ ($\overline{A}$)

Instrucțiunile automatului programabil AP101 care realizează acest operații sunt cele două de mai sus şi ele transferă la adresa locației de memorie sau adresa modulelor de intrare, ieşire, temporizare/contorizare conținutul acumulatorului sau conținutul complementat.

Simultan cu înscrierea unui bistabil dintr-un canal de ieşire se face şi înscrierea locației din memoria RAM, care păstrează imagine a canalului respectiv. Se creează în acest fel o copie reactualizată a stării semnalului de ieşire din automat cu dublu scop:

- se creează posibilitatea implementării de automate finite secvențiale la care există reacții de la ieşiri către intrări, lucru nerealizabil fără această imagine a ieşirii întrucât automatol programabil nu poate sa-şi citească efectiv ieşirile.
- sunt protejate comenzile către proces împotriva semnalelor parazite ce se pot induce în circuitele basculante bistabile asociate fiecărui canal de ieşire. Astfel ieşirile modificate pe durata unui ciclu program sunt înscrise simultan în zonele E şi M, iar ieşirile nemodificate sunt citite la sfârşitul aceluiaşi ciclu program din memoria RAM, care este special protejată împotriva perturbațiilor şi a căderii accidentale a tensiunii de alimentare.

Cele două instrucțiuni 3 şi 4 sunt absolute şi se execută independent de conținutul registrului acumulator ce păstrează rezultatul parțial curent. Există însă şi instrucțiunide transfer de date a căror execuție este condiționată de conținutul registrului acumulator. Acestea sunt:

5. STC adr p (adr p) ⟵ 1
6. RTC adr p (adr p) ⟵ 0

} dacă (A) =1

Acestea realizează setarea unui bistabildin zona de ieşire sau de temporizare sau a unei locații din memoria RAM, condiționat de valoarea rezultatului parțial curent din acumulator.

Prelucrarea logică a datelor

Setul de instrucțiuni al unui automat programabil cu prelucrare la nivel de bit face posibilă implementarea funcțiilor logice exprimate în forme canonice, respectiv forme normale, conjunctive sau disjunctive. Pentru aceasta setul de instrucțiuni include un număr de instrucțiuni logice care asigură implementarea unui sistem logic complet de funcții booleene. O operațiune logică poate opera numai asupra acumulatorului, deci executându-se cu un

singur operand sau cu doi operanzi, respectiv acumulatorul şi un operand din blocurile intrare, temporizare, memorie RAM.

Setul de instrucţiuni logice ale automatului programabil AP101, cu operaţiile pe care le execută, sunt:

Instrucţii care operează cu doi operanzi:

7. AND adr p (A)◄— (A) (adr p)
8. ANDC adr p (A)◄— (A)$\overline{\text{(adr p)}}$
9. OR adr p (A)◄— (A)+(adr p)
10. ORC adr p (A)◄— (A)+$\overline{\text{(adr p)}}$
11. XOR adr p (A)◄— (A)$\oplus$ (adr p)

Instrucţiunile care operează cu un singur operand:

12. NOT (A)◄— ($\overline{A}$)
13. CLR (A)◄— 0

La încheierea execuţiei acestor instrucţiuni noul rezultat este supraînscris în acumulator peste vechiul conţinut al acestuia.

Vom prezenta câteva mici programe de simulare cu AP101 a unor circuite logice uzuale.

Funcţia logică ŞI

$f = a \cdot b$

Programul care va simula această funcţie va încărca în acumulator una din variabile şi, utilizând apoi instrucţiunea AND, ori de câte ori va fi nevoie, va realiza funcţia logică ŞI între acumulator şi celelalte variabile.

Variabilele a şi b se aplică la intrările I_1, respectiv I_2, ale automatului, iar rezultatul se va obţine la ieşire

I_1 ◄— a	*LD* I_1	A◄— a
I_2 ◄— b	*AND* I_2	A◄— a•b
E_1◄— f	*STO* E_1	E_1=f=a•b

Programul care simulează funcţia logică SAU este asemănător cu cel prezentat pentru funcţia ŞI, doar că în loc de AND se va folosi OR.

Funcţia logică SAU-NU

$f = \overline{a+b}$

Considerând aceleaşi adrese pentru variabilele de intrare şi ieşire ca şi în cazul precedent, programul se scrie la fel, cu deosebirea că la sfârşit se memorează complementul acumulatorului.

LD I_1	A ⟵ a
OR I_2	A ⟵ a+b
STOC E_1	$E_1=\overline{a+b}$

Pentru simularea funcţiei logice ŞI-NU programul este asemănător, cu menţiunea că se foloseşte instrucţiunea AND.

Funcţia logică ŞI-SAU-NU

$f = \overline{ab+cd+eg}$

Variabilele a, b, c, d, e şi g se aplică la intrările I_1-I_6 , iar rezultatul se va obţine la ieşirea E_1. Programul care va simula această funcţie va utiliza şi o locaţie din memoria RAM; fie M_1 această locaţie

I_1 ⟵ a	*LD* I_1	calculează şi depune ab în M_1
I_2 ⟵ b	*AND* I_2	
	STO M_1	
I_3 ⟵ c	*LD* I_3	calculează cd şi face suma cu ab
I_4 ⟵ d	*AND* I_4	
	OR M_1	
I_5 ⟵ e	*STO* M_1	
I_6 ⟵ g	*LD* I_5	E_1=f
E_1⟵ f	*AND* I_6	
	OR M_1	
	STOC E_1	

Principii de realizare a salturilor

Instrucţiunile de salt permit unităţii centrale să părăsească prelucrarea liniară a programului, tipică implementării de automate combinaţionale sau secvenţiale cu validarea condiţionată a transferurilor către ieşiri, pentru a comanda derularea programului după secvenţa stabilită sau în funcţie de anumite condiţii.

14. JMP adr (NA) ⟵adr
15. JMPC adr (NA) ⟵adr dacă (A)=1
16. NOP