ასემბლის ინსტრუქციები

A ინსტრუქცია:

@[რიცხვი] – 0 [a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14]

a0...a14 არის 15 ბიტიანი მისამართი, რომელსაც მეხსიერებაში ავირჩევთ და შემდეგ ბრძანებებზე შევძლებთ M-ის დაწერით მივწვდეთ ამ ადგილას ჩაწერილ ინფორმაციას. მაგალითად,

@3

აირჩევს მე-3 ადგილს მეხსიერებაში. ამის შემდეგ თუ დავწერთ,

D = M

მაშინ RAM-ის მე-3 რეგისტრში ჩაწერილი ინფორმაცია ჩაიწერება D რეგისტრში.

არსებობს A ინსტრუქციის მეორენაირი გამოყენებაც. როდესაც კომპიუტერს მივცემთ ნებისმიერ JUMP ინსტრუქციას, შემდეგი ბრძანება გაეშვება ROM-ის იმ ინდექსის მქონე რეგისტრიდან, რაც A რეგისტრში წერია.

მაგალითად, განვილიხილოთ Unconditional Jump-ის შემთხვევა:

@0

0 ; JMP

ამის ეფექტი ის იქნება, რომ შემდეგი ინსტრუქცია გაეშვება ROM-ის პირველივე რეგისტრიდან.

A ინსტრუქციის ორობით ჩანაწერში პირველი ბიტი აუცილებლად ნულია, დანარჩენი კი სასურველი რიცხვის ორობით ჩანაწერს განსაზღვრავს.

C ინსტრუქცია:

dst = comp ; jmp – 1 [s] [p] [a0 a1 a2] [m] [o0 o1 o2] [d0 d1 d2] [j0 j1 j2]

ამ ტიპის ინსტრუქცია ბევრად უფრო კომპლექსურია. ტოლობის მარჯვენა მხარეს ვწერთ იმ ოპერაციას, რაც გვინდა რომ ALU-მ გამოთვალოს. ტოლობის მარჯვენა მხარეს ვწერთ იმ რეგისტრებს, რომლებშიც გვინდა რომ ეს შედეგი ჩაიწეროს. ALU-ს შედეგი დარდება 0-ს წერტილ-მძიმის შემდეგ დაწერილი პირობით. განვიხილოთ მაგალითები:

AMDS = D + A ; JGT

D რეგისტრში მყოფ მნიშვნელობას დაემატება A რეგისტრში მყოფი მნიშვნელობა. შემდეგ M, D და A რეგისტრებში (M-ში იგულისხმება RAM-ში იმ რეგისტრის მნიშვნელობა, რომლის ინდექსიც ემთხვეოდა A რეგისტრში მყოფ მნიშვნელობას ახალი მნიშვნელობის მინიჭებამდე) იწერება ახალი მნიშვნელობები. ამის შემდეგ, მიღებული შედეგი (ამ შემთხვევაში D + A) დარდება 0-ს და თუ ეს შედარება ჭეშმარიტია, მაშინ ხტება იმ ინსტრუქციაზე, რომლის ინდექსიც A რეგისტრში წერია. ჩვენს შემთხვევაში გადახტომა შესრულდებოდა მაშინ, თუ JGT (**J**UMP IF **G**REATER **T**HAN) პირობა იქნებოდა ჭეშმარიტი, ანუ D + A მეტი იქნებოდა ნულზე.

სხვა მსგავსი მაგალითებია:

A = A + 1

M = M * D

DM = -1

ASD = D xor A

D = S + 1

და ა.შ.

თუ გვსურს ამ ბრძანებების ორობითში გადაყვანა, თან უნდა გვქონდეს ასემბლერი ან შემდეგი ცხრილები და ვიცოდეთ რომელი ბიტი რას აღნიშნავს.

1 [s] [p] [a0 a1 a2] [m] [o0 o1 o2] [d0 d1 d2] [j0 j1 j2]

C ინსტრუქციის პირველი ბიტი აუცილებლად 1-ია.

s – განსაზღვრავს სრულდება თუ არა სტეკზე ოპერაცია.

p – თუ სტეკ ოპერაციაა, ეს ბიტი განსაზღვრავს push (0) გვინდა თუ pop (1).

a0 a1 a2 – ეს სამი ბიტი განსაზღვრავს რა ოპერატორს ვიყენებთ.

ოპერატორების ცხრილი:

a0	a1	a2	operator
0	0	0	+
0	0	1	-
0	1	0	*
0	1	1	/
1	0	0	NOT
1	0	1	AND
1	1	0	OR
1	1	1	XOR

m – თუ სტეკ ოპერაცია არ სრულდება, განსაზღვრავს გამოთვლის დროს A-ზე ხდება მოქმედება (0) თუ M-ზე (1).

o0 o1 o2 – განსაზღვრავს თუ რა პოზიცია უკავიათ ოპერანდებს. ერთი მხარე შეიძლება ეკავოს D-ს, ხოლო მეორეში არჩევანი უნდა გაკეთდეს A/M/S-ს შორის **m** და **s** ბიტების დახმარებით.

ოპერანდების ცხრილი (* აღნიშნავს ნებისმიერ ვალიდურ ოპერატორს):

o0	o1	o2	operand (m=0 & s=0)	operand (m=1 & s=0)	operand (s=1 & p=1)
0	0	0	* 0	–	–
0	0	1	D * A	D * M	D * S
0	1	0	* A	* M	* S
0	1	1	* D	–	–
1	0	0	A * D	M * D	S * D
1	0	1	* 1	–	–
1	1	0	A * 1	M * 1	S * 1
1	1	1	D * 1	–	–

d0 d1 d2 – ეს ბიტები განსაზღვრავს თუ სად იწერება ALU-ს მიერ გამოთვლილი შედეგი. **d0** არის A-სთვის, **d1** არის M-სთვის, ხოლო **d2** არის D-სთვის. (თუ ამავდროულად სტეკში გვინდა ჩაწერა, (s=1 & p=0) პირობა უნდა შესრულდეს.

დანიშნულების ადგილების ცხრილი:

d0 (A)	d1 (M)	d2 (D)	destination
0	0	0	null
0	0	1	D
0	1	0	M
0	1	1	MD
1	0	0	A
1	0	1	AD
1	1	0	AM
1	1	1	AMD

j0 j1 j2 – ALU-ს გამოთვლილ შედეგს ადარებს 0-ს. შედარების ოპერატორი განსაზღვრულია ამ სამი ბიტით. j0 აღნიშნავს ნაკლებობას, j1 არის ტოლობა, ხოლო j2 არის მეტობა. ამ სამი ბიტის ვარიაციებით შეიძლება სხვა შედარების ოპერატორების მიღებაც.

შედარების ოპერატორების ცხრილი:

j0 (out < 0)	j1 (out = 0)	j2 (out > 0)	mnemonic	effect
0	0	0	null	no jump
0	0	1	JGT	If out > 0 jump
0	1	0	JEQ	If out = 0 jump
0	1	1	JGE	If out ≥ 0 jump
1	0	0	JLT	If out < 0 jump
1	0	1	JNE	If out ≠ 0 jump
1	1	0	JLE	If out ≤ 0 jump
1	1	1	JMP	jump