Intro

I denna katalog finns en datamängd representerad på tre olika sätt.

Den ”råa” dumpen av PROM-innehållet heter multimil.bin och är 4096 bytes stor (exakt så stor som PROM:et är). För en översikt är det ”kanoniska” formatet lämpligt och återfinns i multimil.canonical medan filen multimil.lst innehåller ett format adress för adress lämpligt för handdisassemblering.

Det finns säkert program som kan disassemblera den här filen. Syftet är nu inte att använda sådana FUSK-program utan att få insikt i hur 6502:ans instruktioner och instruktionsformat fungerar.

Alltså fram med papper och penna och en listning över 6502:ans instruktioner och sätt igång och klura. Meddela eventuella framsteg!!!

Exempel på hur man kan göra

För att underlätta starten visar jag här den ungefärliga gången på de första byten i filen, programmet börjar på rad 0:

”78 d8 a2 2f 9a a2 ff 8e 0f 60 8e 00 40 8e 03 40 ...”

Man måste ha en lista över 6502:s kommandon och deras hexadecimala representation framme. Den i databladet (kolla på lästipsen där finns länk till pdf-en) duger. Här ser man vilka kommandon som finns, vad de gör och hur de ser ut i minnet (det minne vi ska ”knäcka”).

Den första byten ”78” avkodas med tabellen som SEI, dvs Set Enable Interrupt. Den har adresseringsmoden implied vilket verkar rimligt. Vi kan nu se att SEI tar 1 byte (ur kolumnen ”#”) och tar två maskincykler att utföra (vilket är ointressant om man inte ska räkna ut hur många mikrosekunder en instruktion tar).

OK, SEI tar en byte, dvs nästa byte är början på en ny instruktion. ”d8” hittar vi i tabellen som CLD, Clear Decimal Mode, en en-bytes instruktion det med, som säger åt processorn att inte behandla tal som BCD-kodade tal.

”a2” måste också vara en ny instruktion då CLD bara ta en byte. ”a2” hittar hittar vi som LDX. Den instruktionen är på två bytes (#-kolumnen är 2). Detta betyder att den har ett enbytes adressfält också, dvs byte nummer två är en adress och ingen instruktion. Adressen är uppenbarligen ”2f”.

Instruktionen därefter är ”9a” dvs TXS, en enbytes instruktion som överför indexregistret X:s värde till Stackregistret S. Man har alltså laddat stackpekaren med ”a2” som X fick i förra instruktionen.

Sedan kommer en ”a2” igen, dvs LDX, dock ska LDX nu laddas med ”ff”.

”8e” är ånyo ett kommando, STX, Store Index X in Memory. Vi ser att den tar tre bytes, det betyder att den har ett 2-bytes, dvs 16-bitars argument. ”0f 60” är alltså någon adress

i minnet där X ska lagras. Om ”0f 60” ska tolkas som det hexadecimala talet $0f60 eller som $600f vet vi inte än. Det får vi klura ut med ledning av framtiden...

Härnäst kommer ytterligare ”8e”, STX-kommando, faktiskt två efter varann, med respektive argument ”00 40” och ”03 40”. Om dessa ska tolkas till adresser verkar det väl rimligt om de tolkas som $4000 respektive $4003, då hamnar de nära varann. Man kan naturligtvis tänka sig att man vill placera ut X:s värde på helt skilda delar av minnet, men det är inte sannolikt. Framtiden får utvisa.

I och med detta kan vi gå tillbaks till den tidigare osäkra adressen ”0f 60” och kalla den för $600f. Att alla adresser nu verkar ligga i närheten av ”N000” är ytterligare något som stärker misstanken att vi tolkar adresserna på rätt sätt.

Och sen är det bara att klura på. Det tar tid men ger en oöverträffad erfarenhet.

Koden som vi knäckt hittills skulle alltså kunna skrivas som

Programrad	Instruktion	kommentar
000	SEI	; förhindra vidare avbrott
	CLD	; Inga BCD-tal förekommer. (Det tackar vi speciellt för)
	LDX $2f	; Ladda Stackpekaren med adressen $2f.
	TXS	
	LDX $ff	; Ladda konstanten $ff, dvs -1
	STX $600f	; $600f initieras till $ff
	STX $4000	; samma för $4000...
	STX $4003	; ... och $4003
	:	
	:	

Det verkar som att programmet börjar med att initialisera omgivningen. Man måste komma ihåg att det sitter hårdvara (kretsen till vänster på bilden som visar var minnet satt, till exempel) *minnesmappat*, dvs om man skriver till vissa adresser (okänt vilka just nu) så sker en skrivning till yttre kretsar istället. Sådana kretsar brukar ligga på adresser som är lätta att avkoda med enkel digital logik (and/or-grindar osv). Så både $6000, $4000 och $4003 är kandidater till extern hårdvara. Det behöver inte vara just så, det kanske kan komma fram senare.

Det kan vara idé att inte börja disassemblera direkt på rad noll. Initieringen skvallrar en del om hur systemet är upplagt, men sedan kanske det finns intressanta rutiner som känner av tangentbord, omvandlar binärtal till siffror (leta efter konstanterna 1, 10, 100 omedelbart nära varann) att visa upp på displayen, drivrutiner till displayen m.m. som betyder mer för sammanhanget?

Fortsätt så länge det är roligt!

/Micke

Adresseringsmoder

Processorn stöder ett antal adresseringsmoder. De finns angivna i databladet men där står inte hur de skrivs. Så här fungerar det:

Adresseringsmod	Skrivsätt
Immediate	#aa
Absolute	aaaa
Zero page	aa
Implied	
Indirect Absolute	(aaaa)
Absolute Indexed, X	aaaa,X
Absolute Indexed, Y	aaaa,Y
Zero page Indexed, X	aa,X
Zero page Indexed, Y	aa,Y
Indexed Indirect	(aa,X)
Indirect Indexed	(aa),Y
Relative	aa eller aaaa
Ackumulator	A

Objektkod

Man kan roa sig med att studera hur de olika hexkoderna hänger ihop också. Det kan avslöja en del av hur processorn ser ut internt, dvs vilka bitar i instruktionen som styr vad osv. Tabellerna nedan får tala för sig själva:

Fält	Betydelse	
aaa		
	000	(adr,X)
	001	adr
	010	data
	011	adr16
	100	(adr),Y
	101	adr,X
	110	adr16,Y
	111	adr16,X
bb		
	00	adr
	01	adr16
	10	adr,X
	11	adr16,X
bbb		
	001	adr
	010	ackumulator
	011	adr16
	101	adr,X
	111	adr16,X
cc		
	00	data
	01	adr
	11	adr16
ddd		
	000	data
	001	adr
	011	adr16
	101	adr,Y i LDX / adr,X i LDY
	111	adr16,Y i LDX / adr16,X i LDY
pp		Andra byten i en två- eller trebytesinstruktion
qq		Tredje byten i en trebytesinstruktion
x		Bit som väljer adressmod

Mnemonic	Operand(s)	Object code	Bytes
ADC		011aaa01	
	data or a8	pp	2
	a16	qq	3
AND		001aaa01	
	data or a8	pp	2
	a16	qq	3
ASL	A	000bbb10	1
	adr or adr,X	pp	2
	adr16 or adr16,X	qq	3
	:		
	:		

Resten får ni fylla i själva — eller om det är en allmän önskan kan jag skriva den kompletta listan senare.