Per lez. 15/12/2010

ricapitoliamo quanto fatto su "ADDRESSING MODES E CODIFICA DELLE ISTRUZIONI"

1 abbiamo cominciato 2 lez. fa studiando l'esecuzione del task

```
MUL   R1.L, R2.H, R3
```

dove nel controllo gli operandi erano non flessibili: la mult. cioè funzionava solo per questi 3 : R1.L, R2.H, R3.
In sostanza, era una MACCHINA DEDICATA realizzata con L'HARDWARE DI PARTE OPERATIVA DI UNA CPU A SINGOLO BUS INTERNO.
Avremmo potuto ~~[illegible]~~ il task col solo mnemonico "MUL"!

2 Abbiamo allargato il discorso considerando

```
MUL   reg1, reg2, reg3
```

che utilizza il register file e codifica gli operandi.
Per far questo abbiamo anche cambiato l'architettura, naturalmente, PRIMO: inserendo la gestione del register file, SECONDO: rendendo il calcolo indipendente dall'uso di registri VISIBILI AL PROGRAMMATORE: si sono usati cioè tutti REGISTRI TEMPORANEI (tipo ACC). La codifica di MUL R1.L, R2.H, R3 era ha in più rispetto a prima i 3 + 3 + 2 campi relativi alla codifica operandi — che però sono tutti registri, e quindi non necessitano di altre info per essere completamente decifrati.

3 Ancora più in generale, abbiamo notato che l'esecuzione della MUL (come in generale di gli istruzioni) può ritenersi strutturata in

LOAD OPERANDI → CALCOLO → STORE RISULTATI

e quindi abbiamo ulteriormente generalizzato il discorso considerando

MUL op1, op2, op3

dove stavolta op_i può essere specificato con 1 di 4 ADDRESSING MODES:

- registro (come sopra)
- diretto di memoria
- indiretto di registro
- immediato

(qui ho fatto esempi di sintassi)

Note: (1) op3 non può essere immediato;

(2) nel caso indiretto di registro, nel registro c'è un indirizzo, e quindi il bus interno deve avere una dimensione sufficiente per esso (e questo è evidente, anche perché altri indirizzi, es. PC, passano dal bus interno), e molti dati e indirizzi nel caso più semplice fatto a lez. hanno le stesse dimensioni (ho detto 16 bit).

(3) Non è necessario specificare nella codifica dell'istr. che op1 e op2 sono da 8 bit, e op3 a 16 bit (controllo word/byte per il register file), perché la MUL è strutturata SEMPRE così.

La codifica di ciascun operando può prendere

- reg. — 3 bit per op1 e op2, 2 bit per op3
- dir — 16 bit
- ind. reg. — 2 bit (si usa la versione interna Ri)
- imm. — 8 bit (solo op1 e op2)

Inoltre, bisogneri CODIFICARE I MODI DI INDIRIZZAMENTO, dunque

2 bit per operando (essendo 4 gli ADD. MODES).

La lunghezza MAX e MIN dell'ISA è dunque

min: $size(OPCODE) + \underbrace{2+2+2}_{\text{codifica ADDM}} + \overbrace{\underbrace{2}_{\text{ind. reg.}} + \underbrace{2}_{\text{ind. reg.}} + \underbrace{2}_{\text{reg.}}}^{\text{operandi}} = \underbrace{size(OPCODE) + 6}_{\text{base: c'è sempre}} + 6$

max: $size(OPCODE) + \underbrace{2+2+2}_{} + \underbrace{16+16+16}_{\text{tutti e 3 dir.}} = size(OPCODE) + 6 + 48$

Ipotizzando un bus dati a 16 bit, e $size(OPCODE) = 6$ bit, (come fatto)

abbiamo

$\lceil 18/16 \rceil$

min instr. length = 18 bit $\Longrightarrow$ 2 cicli di bus

max " " = 60 bit $\Longrightarrow$ 4 "

Qui ha detto che NEANCHE I CISC SI POSSONO IN GENERE PERMETTERE UNA TALE FLESSIBILITA' nel NUMERO e nel TIPO di operandi. Es 8086:

- al max 2 operandi (quindi la dest o è IMPLICITA o è una delle due sorgenti)
- di cui 1 solo può essere di memoria: l'altro dev'essere di registro

NOTA: la seconda clausola non consente neanche che uno dei due operandi sia diretto e l'altro indiretto di registro: in questo caso la lunghezza di codifica istr. sarebbe ok, ma l'esecuzione si allungherebbe (fase di load e/o store op.)

4 Abbiamo quindi studiato (anche il controllo) di

| MUL op1, reg2, reg3 |

op1 ∈ {reg, dir, ind reg, imm}

che ha codifica

OPCODE	ADDM1	op1	reg2	reg3
6	2	variabile: 3, 16, 2, 8	3	2

Tabella codifica op1:

ADDM1	modo	# bit	campi IR: op1 (var.)	reg2 (3)	reg3 (2)	Lunghezza istruzione
0 0	registro	3	IR[8 – 10]	IR[11 – 13]	IR[14 – 15]	16
0 1	diretto	16	IR[8 – 23]	IR[24 – 26]	IR[27 – 28]	29
1 0	indir. reg.	2	IR[8 – 9]	IR[10 – 12]	IR[13 – 14]	15
1 1	immediato	8	IR[8 – 15]	IR[16 – 18]	IR[19 – 20]	21

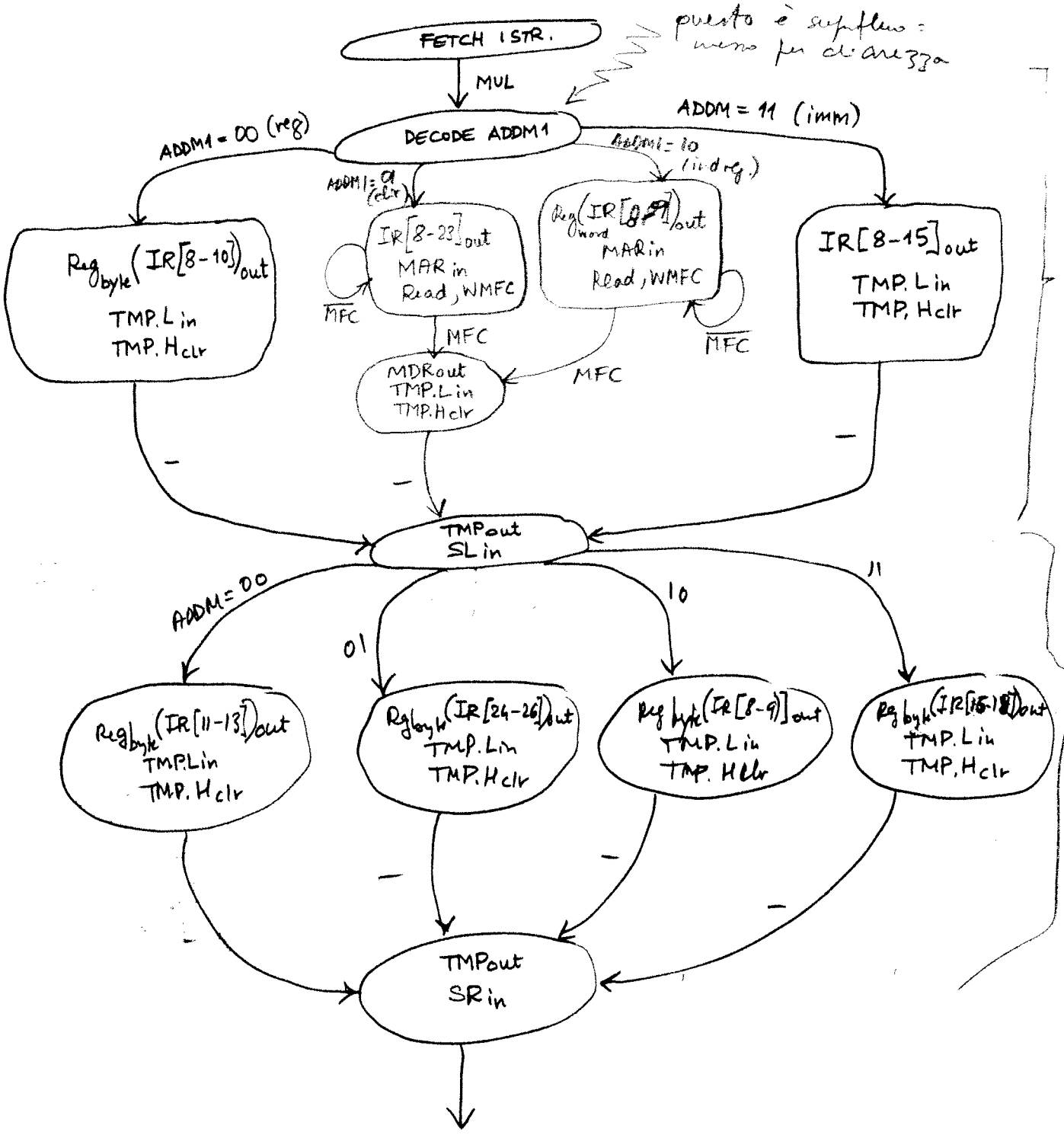
(# cicli macchina)

Quindi la min length è 15 bit, la max è 29;
inoltre il # di cicli di bus necessari è risp. 1, 2, 1, 2,
molto ridotto rispetto a prima.

☆
4 + Ncompute + 1 (tralasciando stati cascheti)
5 + Nw + Ncompute + 1
/
4 + N[illegible] + 1

Controllo: %

CALCOLARE POI IL # DI CICLI MACCHINA NECESSARI ∀ ADDM1


FETCH ISTR.
questo è superfluo: messo per chiarezza
MUL
DECODE ADDM1
ADDM1 = 00 (reg)
ADDM1 = 01 (dir)
ADDM1 = 10 (ind reg.)
ADDM = 11 (imm)
Reg byte (IR[8-10])out
TMP.L in
TMP.H clr
IR[8-23]out
MAR in
Read, WMFC
MFC
Reg word (IR[8-23])out
MAR in
Read, WMFC
MFC
IR[8-15]out
TMP.L in
TMP.H clr
MDRout
TMP.L in
TMP.H clr
MFC
TMPout
SL in
ADDM=00
01
10
11
Reg byte (IR[11-13])out
TMP.L in
TMP.H clr
Reg byte (IR[24-26])out
TMP.L in
TMP.H clr
Reg byte (IR[8-9])out
TMP.L in
TMP.H clr
Reg byte (IR[16-18])out
TMP.L in
TMP.H clr
TMPout
SR in
Load operandi
alla fase di calcolo

Rivediamo per bene la fase di calcolo: