Corso di calcolatori elettronici I per Ingegneria Informatica a.a. 2004-2005
(pagina web: www.dsi.unifi.it/users/colombo/calc0405.html)

Carlo Colombo (tel. 0554796329, e-mail colombo@dsi.unifi.it)

GIORNALE DI BORDO

I.1/1-3 - *Martedì 22 febbraio 2005 (3 ore)*

Presentazione generale del corso. Le lezioni cominceranno sempre alle 8:45. Martedì e Giovedì 3 ore, Mercoledì 2 ore. Ricevimento in aula 24, dopo la lezione del mercoledì. C'è un nuovo libro di testo. Quest'anno approfondiremo maggiormente la parte di programmazione. Premio per chi consegna, a fine corso, appunti insieme ordinati e completi. Contenuti del corso: introduzione alle architetture dei calcolatori elettronici. Differenza tra architettura (=ciò che deve conoscere il programmatore Assembly) e organizzazione (=una realizzazione particolare di un'architettura). Calcolatori, o meglio sistemi di elaborazione dell'informazione. Manipolazione di simboli. 1° esempio: rappresentazione di una macchina a stati finiti: la macchina che eroga il caffè, accettando monete da 0.10, 0.20 e 0.50 € (costo del caffè: 0.50 €). Ad ingressi uguali corrispondono uscite differenti: memoria (degli ingressi precedenti). Il salto tra due stati può essere obbligato, o condizionato al valore di un ingresso. In un sistema di elaborazione, tutto (ingressi, stati, uscite) è rappresentato per mezzo di due soli simboli: convenzionalmente 0 e 1. Con n bit si possono rappresentare fino a 2^n configurazioni distinte. Potenze del due importanti in informatica (es. 1K=2^{10}=1024 ≠ 1000, con 1M=1Kx1K, etc.). Tabella delle potenze da 2^0 a 2^{10}, e calcolo di ogni altra potenza con la proprietà dell'esponente: $2^{(h+k)} = 2^h \times 2^k$. Bit, byte, word. *La torre di Hanoi con N dischi*: *soluzione in 2^N-1 passi.* La legge di Moore: le prestazioni dei calcolatori raddoppiano ogni 18 mesi. Testo, numeri, immagini, suoni sono rappresentabili utilizzando numeri interi. Anche i numeri in virgola mobile sono rappresentabili da una coppia di interi. Sistemi digitali: transistor usato come interruttore, con due possibili stati: acceso (1) o spento (0) – gli stati logici sono associati a quelli elettrici secondo una convenzione arbitraria. 2° esempio: macchina di Turing che addiziona interi. Nastro infinito, simboli da un alfabeto finito, testina di lettura/scrittura, stati (finiti). Turing ebbe l'idea di specificare (rappresentare) il funzionamento stesso di una macchina specializzata in un dato compito (ad es. addizione) coi simboli di un alfabeto finito, definendo così una macchina universale, che accetta come ingressi descrizioni di macchine specializzate. Si arriva così al concetto fondamentale di programma memorizzato. Anche le istruzioni del programma sono rappresentate attraverso numeri interi espressi in notazione binaria. 3° esempio: preparazione di una mousse al cioccolato. Rappresentazione schematica dell'algoritmo di preparazione della mousse. Il grado di parallelismo dipende dalle risorse disponibili (eventuali "sotto-cuochi"). Ingredienti=dati in ingresso, ricetta=programma memorizzato, Unità Operativa (UO)=cucina, utensili ed eventuali collaboratori dello chef, uscita=mousse. Chef=Unità di Controllo (UC): comandi (per la UO) e condizioni (dalla UO). Se lo chef sapesse preparare solo questo piatto, la macchina sarebbe dedicata e sarebbe superfluo fornirgli la ricetta. La ricetta va fornita in un linguaggio (=linguaggio macchina) comprensibile alla UC. Il corso studia la relazione tra hardware e software in un sistema di elaborazione. Non studieremo il transistor, parleremo invece dei "mattoncini" fondamentali dei sistemi logici (porte, flip-flop) e di alcuni dei loro arrangiamenti più tipici (multiplexer, registri). Le reti logiche sono combinatorie (senza memoria) o sequenziali (con memoria): entrambi i tipi sono importanti nel calcolatore, che ha un comportamento complessivo di tipo sequenziale, in quanto dipendente non solo dall'ingresso attuale ma dalla storia degli ingressi. Architettura "di" Von Neumann: il programma è memorizzato nello stesso luogo che contiene anche i dati da elaborare. Input, output, unità operativa, unità di controllo, memoria. La comunicazione tra I/O, CPU e MEM avviene attraverso 3 distinti canali (o bus): dati, indirizzi, controlli. La memoria colloquia bidirezionalmente col processore ricevendone i dati e permettendo alla CPU di "scaricare" un po' alla volta le istruzioni in linguaggio macchina (assembly), le quali indicano come operare sui dati. Colloquio CPU-memoria: se il bus indirizzi è di m bit, lo spazio di indirizzamento (numero di parole distinte generabili) è 2^m. La memoria è come una cassettiera,

dove ognuno dei 2^m cassetti contiene un oggetto di dimensione standard (=1 parola da *n* bit). Dimensione complessiva (in bit) di una memoria: 2^m (words) x *n* (bit/word). Memory size in byte, in Mbyte. Segnale di controllo *WRITE* (=scrivi): asserito (ossia "fa ciò che dice il suo nome") se alto, se basso invece significa "leggi" (scrivere e leggere sono azioni mutuamente esclusive). Il corso si compone di otto moduli e tre parti: [I] (1) Introduzione ai sistemi digitali, (2) Rappresentazione di dati: interi con segno e caratteri alfanumerici (la rappresentazione delle istruzioni, non standard, sarà trattata nelle parti II.4 e III.7), (3) Reti logiche; [II] (4) CPU, o Microprocessore, (5) Memoria, (6) Input-output, o I/O; [III] (7) Hardware 8086, (8) Software 8086. (L'8086 è l'antenato del Pentium: ha meno funzioni, ed è più semplice da studiare e da programmare.)

I.2/4-5 - *Mercoledì 23 febbraio 2005 (2 ore)*

Rappresentazione di interi positivi. Codifica posizionale: base *B*, alfabeto di *B* cifre, che agiscono come pesi per potenze di *B*. Parole di *k* cifre consentono di rappresentare fino a B^k numeri incluso lo 0. Range di rappresentazione: [0, B^k -1]. Rappresentazioni decimale, binaria, esadecimale. Più grande la base, più corti i numeri. LSB e MSB. Moltiplicazione e divisione intera per 2 come scorrimento (shift). Conversione da un sistema all'altro: da B^k a *B* e viceversa, da *X* a *Y* (esempio: *X*=10 e *Y*=2). Esercizi: convertire il decimale 2710 in base due. Interi con segno: rappresentazione in: (0) modulo e segno, (1) complemento a uno, (2) complemento a due. Il MSB a 1 indica numero negativo. Complemento a 1 e complemento a 2 come somme di potenze. Il range di rappresentabilità dei numeri positivi si dimezza. Determinazione del minimo numero di bit per rappresentare un numero dato. Attenzione: per rappresentare +27 in complemento non bastano 5 bit: ce ne vogliono 6. Il complemento a due consente di rappresentare lo 0 in un solo modo. Uno stesso numerale dà luogo a numeri diversi a seconda della rappresentazione utilizzata. Se il MSB è 0 (numero non negativo), allora l'interpretazione è identica in tutte e quattro le rappresentazioni viste.

I.3/6-8 - *Giovedì 24 febbraio 2005 (3 ore)*

Fare il complemento di un numero equivale a cambiarlo di segno. Complemento a uno = complementazione logica dei bit. L'operatore NOT: tavola di verità (Wittgenstein), simbolo grafico, notazione algebrica. Complemento a due = complemento a uno + 1. Complemento a due veloce. Attenzione: interpretare un numerale secondo la rappresentazione in complemento a *x* non vuol dire "fare il complemento a *x*", che significa invece cambiare di segno un numerale che sia già rappresentato in complemento a *x*. Utilità delle varie forme di rappresentazione: modulo e segno impiegato nel trattamento dei numeri in virgola mobile, ma richiede circuiti di addizione e di sottrazione. Con le rappresentazioni in complemento si usano solo addizionatori, ma in complemento a uno bisogna sommare l'eventuale carry. Addizione binaria di due bit (Leibniz). Rappresentazione grafica dell'addizione di parole di *k* bit. Rappresentazione naturale: carry = overflow. Complemento a due: carry ≠ overflow. Condizione di overflow e suo rilevamento: la funzione logica OVERFLOW(a,b,s), rappresentata secondo la tavola di verità e la somma di prodotti. Operatori AND e OR a *n* ingressi. Abuso di notazione: "+" e "•" indicano somma e prodotto logico. Proposti tre esercizi dai compiti d'esame: interpretazione di un numerale secondo vari sistemi di numerazione (07/02/03), complementi a uno e a due (24/09/03), sommatore a due bit con uscite di carry e overflow (26/04/02). Estensione del segno: serve a poter sommare numerali di lunghezza diversa. Informatica e crittografia. Rappresentazione di testi: *il metodo di cifratura di Francesco Bacone*. Standard ASCII. Attenzione: '7' ≠ 7. Come passare da '7' a 7: metodo aritmetico (sottrazione di '0') e metodo logico (AND con maschera 0Fh). Standard BCD (da fare sul libro).

I.4/9-11 – *Mercoledì 2 marzo 2005 (3 ore)*

Dopo i contributi di Leibniz ('600), la logica passa definitivamente dalla filosofia alla matematica nell'800 con Boole. Logica proposizionale e dei predicati: entrambe coerenti e complete. Indecidibilità della logica dei predicati (Turing 1936). Incompletezza dell'aritmetica (Gödel 1931). Algebra booleana: è un apparato formale che può essere interpretato in molti modi diversi (es. logica proposizionale (Post 1921), reti logiche (Shannon 1938)). *La semplicità "imbarazzante" (Odifreddi) del calcolo dei sillogismi usando l'algebra booleana*. Costanti, variabili, espressioni, funzioni. Enumerazione di funzioni: il caso di uno, due ed *n* ingressi. Funzioni notevoli di due ingressi: AND, OR (inclusivo, o "vel"), XOR (OR esclusivo, o "aut…aut"), NAND, NOR, $x \leq y$ (o implicazione materiale $x \rightarrow y$, *o* "se *x*, allora *y*"), etc. Espressione generale di una funzione come somma (logica) pesata di prodotti (logici) fondamentali. Il multiplexer (MUX, o selettore) e la rete logica corrispondente. Proprietà dell'algebra booleana e principio di dualità (una stessa proprietà continua a valere se si scambiano tra loro le costanti 0 e 1, e gli operatori + e •). Alcune proprietà non sono valide nell'algebra ordinaria. Elementi neutri e forzanti, e loro uso nelle maschere. Le leggi di De Morgan. Rappresentazioni canoniche di una funzione booleana: somma di prodotti e prodotto di somme. Semplificazione di espressioni: equivalenza logica, ma diversità di realizzazione circuitale. NAND e NOR operatori (o connettivi) funzionalmente completi. Realizzazione delle rappresentazioni canoniche: SP con soli NAND, PS con soli NOR.

I.5/12-14 – *Giovedì 3 marzo 2005 (3 ore)*

Esercizio d'esame (b) del 20/11/03: realizzazione di una funzione di tre variabili con sole porte NAND. Molte le soluzioni possibili: differenza tra la specifica logica di una funzione e la sua realizzazione circuitale. Come passare da una forma non canonica ad una forma canonica. *Verifica o refutazione di una tautologia: il "calculemus!" di Leibniz possibile con i circuiti logici.* XOR: simbolo grafico, realizzazione con quattro NAND a due ingressi. L'aritmetica finita possibile attraverso la logica: il sommatore a *n* bit. Half adder. Full adder. Proprietà dell'XOR a *n* ingressi: il risultato non cambia se si complementa un numero pari di variabili. Sommatore "ripple carry": il ritardo è nT,con T ritardo sul singolo full adder. Demultiplexer. Decoder binario ($n \rightarrow 2^n$). Decoder di configurazioni binarie. Esercizio: decodifica di indirizzi di I/O (3 casi: F9h e FAh; FAh e FBh; da F8h a FBh). XOR per la generazione e la rilevazione di parità.

I.6/15-17 – *Martedì 8 marzo 2005 (3 ore)*

Circuiti di parità come caso particolare di codifica di canale. Si aggiunge ridondanza in modo controllato. Il parity checker individua la presenza di errori di parità, ma non li localizza, e non può eliminarli. Canale, rumore. A differenza dei segnali analogici, quelli digitali sono ricevuti perfettamente esatti o perfettamente sbagliati. Codici a correzione d'errore. La trasmissione ripetuta consente la correzione di errori, fa calare la probabilità d'errore ma fa aumentare tempi e costi. Il calcolo delle probabilità come ulteriore interpretazione dell'algebra di Boole. Teoria dell'informazione (Shannon – ancora lui – 1948). La quantità di informazione in un messaggio tratto da un alfabeto di n simboli equiprobabili. L'alfabeto inglese, le 10 cifre numeriche. Il numero medio di bit trasmessi cambia con la codifica adottata. La codifica migliore è quella che invia un numero di bit medio pari alla quantità di informazione. La codifica uniforme non è la migliore. Il caso generale: simboli non equiprobabili (ETAOINSHRDLU, legge di Benford per la prima cifra di un numero). La quantità di informazione è legata all'improbabilità di un messaggio. Simboli frequenti devono avere codifiche corte (vedere ad es. le congiunzioni nel linguaggio naturale). Ridondanza. Meccanismo "a fisarmonica": la codifica di sorgente comprime i dati, eliminando la ridondanza in un messaggio prima che questo sia ricodificato (con espansione) per il canale. Il caso del CD audio. Comparatore. Codificatore binario. Codificatore con priorità (don't care sugli ingressi): l'esempio della tastiera del computer. Codificatore BCD-7 segmenti (è del tipo $4 \rightarrow 7$): si sfruttano i don't care sulle uscite per ottenere circuiti compatti. Una qualsiasi funzione combinatoria con *n* ingressi ed *m* uscite realizzata come somma di prodotti può riguardarsi come la cascata di

un decodificatore binario $n \rightarrow 2^n$ (matrice AND) e di un codificatore $2^n \rightarrow m$ (matrice OR). ROM (=read only memory): realizzano la tavola di verità di una qualsiasi funzione combinatoria. PROM, EPROM. La ROM size raddoppia se aggiungo un ingresso. ROM e macchinette calcolatrici: si precalcolano tutti i possibili risultati (differenza tra cibi precotti e cucinati espressi). Memoria vs calcolo. Una semplice ALU (vedere libro): comandi e condizioni da e per l'unità di controllo.

I.7/18-20 – *mercoledì 9 marzo 2005 (3 ore)*

Latch SR: realizzazione con porte NOR. Dimostrazione della natura sequenziale del circuito: a SR=00 corrispondono due possibili uscite. A parità di elementi logici, è il tipo di connessione a fare la differenza. La memoria è dovuta alla presenza di un anello (*cfr. le "mani che si disegnano" di Escher, e il libro di Hofstadter*). Analisi del latch SR: ricerca delle configurazioni stabili: hold (*SR*=00), reset (*SR*=01), set (*SR*=10). La configurazione "proibita" SR=11 dà luogo ad uno stato stabile, dal quale però si raggiunge uno stato ignoto o instabile nel caso di commutazione simultanea degli ingressi 11→00. Flip-flop SR: basta aggiungere il clock. Macchine sincrone e asincrone. Nelle macchine sincrone, un controllo semplice si paga in termini di velocità di elaborazione. Campionamento sui livelli e sui fronti (di salita o di discesa). Tabella delle transizioni, equazione caratteristica (ponendo a 1 i don't care sulla configurazione proibita), diagramma degli stati, diagramma temporale. Esercizio proposto: facendo uso dell'equivalenza NOR-NAND, disegnare il latch SR e il FF SR con sole porte NAND. Tutti i FF hanno due stati, hanno per funzione d'uscita l'identità, e si caratterizzano per l'equazione di transizione dello stato. Il FF JK: si comporta come un SR, con in più la modalità toggle (commutazione di stato), corrispondente alla configurazione d'ingresso *JK*=11, qui permessa. Il FF D ("delay") ottenibile ponendo $D=J=\underline{K}$ (questo, tra l'altro, evita la configurazione proibita). Il FF D non ha le modalità hold e toggle del JK. Il FF T ("toggle"): ottenuto con *T*=*J*=*K*, può lavorare in modalità hold (*T*=0) o toggle (*T*=1). Equazione caratteristica del FF D: non c'è dipendenza dallo stato (è l'unico caso, tra tutti i FF). Altre rappresentazioni di macchine sequenziali: schema a blocchi (*f*-M-*g*, con M memoria dello stato), equazioni caratteristiche ($y'=f(x,y)$, $z=g(x,y)$). Macchine di Moore: $z=g(y)$. Tutti i FF hanno un automa a due stati, e quindi memorizzano un bit. Le differenze sono nella funzione *f*.

I.8/21-23 – *giovedì 10 marzo 2005 (3 ore)*

Sommatore seriale. Suo comportamento sequenziale. Tabella di flusso. Codifica binaria e tabella delle transizioni. La codifica influenza la realizzazione. 1ª codifica: la parte combinatoria è un full adder. 2ª codifica: al full adder si aggiungono due negatori. Confronto con lo schema a blocchi *f,M,g*. Realizzazione della tabella delle transizioni con ROM + registro di stato: generalizza la realizzazione della tabella di verità di una funzione combinatoria con ROM vista due lezioni fa. Come nell'altro caso, la realizzazione è comoda ma dispendiosa (nel caso sequenziale, tale realizzazione consente di realizzare un qualunque automa, e quindi è fatta in modo che ogni passaggio di stato sia un "salto" governato dall'ingresso). Il progetto logico è in buona parte legato alla modellazione del problema da risolvere: il resto lo fa il computer (CAD). (Cfr. esercizio d'esame col problema dei due recipienti) Realizzazione con topologia di Mealy e di Moore. Il blocco *M* (registro di stato) è composto da una batteria di FF D. Sommatore seriale in versione Moore. Esercizi d'esame consigliati: analisi e rappresentazione di macchine sequenziali. Riepilogo delle rappresentazioni viste finora: equazioni caratteristiche, diagramma degli stati, tabella delle transizioni, schema a blocchi, realizzazioni con ROM. Altra rappresentazione: parte operativa/parte di controllo. Sono entrambe in generale macchine sequenziali. Ognuna è in generale realizzata a suo modo. Tipicamente, la parte di controllo si realizza in modo standard con ROM + registro di stato (o con varianti, che vedremo in seguito, che useranno contatori per sfruttare la codifica incrementale degli stati; anche la ROM può essere rimpiazzata con PLA o PAL), una volta che si sia definita la parte operativa sulla base del funzionamento logico richiesto. Sommatore seriale: la PO è il full adder (con eventuali negatori), la PC è il FF D (di cui abbiamo disegnato l'automa). *Fino a quanto conta una gallina?* Il contatore come rete autonoma (no ingressi, no salti). Il conteggio è modulo *M*. Gli stati effettivi della macchina possono essere meno di 2^n, con *n* numero di FF. Con

un ingresso esterno (UP/DOWN) si conta in avanti e all'indietro (disegnato l'automa corrispondente). Realizzazione di un contatore up con FF T. Lo stesso contatore si può realizzare con FF D, e con una diversa sezione combinatoria di contorno. Lo stesso FF T si può costruire con un XOR e un FF D.

I.9/24-26 – *martedì 15 marzo 2005 (3 ore)*

Realizzazione di un contatore down con FF T. Introduzione dell'ingresso UP/DOWN, e scrittura delle equazioni di un contatore up/down con FF T. Generatori di sequenze: fanno uso di un contatore con in cascata un blocco combinatorio (funzione *g*). Esempio: realizzazione di un contatore down basata su un contatore up. Il registro a scorrimento: modalità hold, shift left, shift right, load (caricamento parallelo). Il registro dati semplice ha solo l'hold e il load. Ingressi asincroni per i registri: preset, clear. Uso dell'ingresso load di un contatore up per la realizzazione di un contatore up/down: si pone load= UP; in modalità di conteggio down, si carica in parallelo lo stato futuro generato con un blocco combinatorio a monte del contatore (funzione *f*). Il contatore in modalità load si comporta come un semplice registro di stato. Sfruttamento della codifica incrementale degli stati: la realizzazione è meno generale, ma meno dispendiosa, perché più aderente al comportamento logico da progettare. Esempio della diramazione singola (governata da un ingresso *x*). L'automa usato nell'esempio ha un altro salto, non legato ad una diramazione (e quindi ad un ingresso), ma dovuto ad una codifica non incrementale tra due stati successivi. Realizzazione dell'automa: basata su contatore con ingresso di load + generazione dello stato futuro (solo negli stati di salto) + funzione g (per la generazione delle uscite *z* e delle uscite ausiliarie "load" e "numero del salto"). Codificando solo il numero del salto e non lo stato si risparmiano bit in ingresso al blocco *f*. *Perché i corvi sanno contare fino a quattro. Il prete matematico che codifica in binario l'ora e la suona con due campane dal suono diverso per fare sapere anche ai corvi l'ora esatta*. Moltiplicazione sequenziale. Moltiplicazione binaria a mano: si ricopia il moltiplicando *M* scalato a sinistra di *k* bit se il *k*-mo bit del moltiplicatore *m* è 1, e si somma il tutto. Il prodotto *P* di due numeri di *n* bit sta su *2n* bit. Il risultato è ottenibile attraverso *n* somme successive (prodotti parziali). Sfruttando la codifica posizionale dei numeri, si evita di sommare *m* volte il numero *M* (*cfr. ruota di Leibniz*), e si ottiene il risultato dopo un numero fisso di passi. Ad ogni somma, il prodotto parziale cresce di una cifra binaria, e si ottiene una nuova cifra definitiva del prodotto. Inizializzazione: *c=A=0, Q=m* [*P=A:Q*]; l'add *cA←M+A* non si fa sempre, lo shift a destra di *cP* sì. Schema della parte operativa del moltiplicatore sequenziale. Segnali di controllo: *load* (per *c* e *A*), *clear* (per *c* e *A*), shiftR (per *c* e *P*), *add* (per il MPX). Condizioni: l'LSB di *P*.

I.10/27-29 – *mercoledì 16 marzo 2005 (3 ore)*

Critiche allo schema della parte operativa della lezione precedente: (1) mancava il carry-in a 0 sulla ALU; (2) per far lavorare la ALU come sommatore, bisogna mantenere costantemente asserito il segnale di selezione di modo "add"; (3) bisogna prevedere altri segnali di controllo utili nella fase di inizializzazione: LOAD (caricamento parallelo degli operandi in M e Q), CLEAR (azzeramento dei registri c e A); (4) serve un controllo LOADA per effettuare il caricamento parallelo in A della somma calcolata dalla ALU ad ogni passo: tale segnale dovrà essere generato sul fronte di discesa del clock, in modo da garantire ingressi stabili all'ingresso della ALU per tutta la fase alta del clock. Algoritmo di moltiplicazione: realizzabile anche con rete puramente combinatoria. Moltiplicazione seriale: diversamente dalla somma seriale, è irrealizzabile con ASF. Flowchart dell'algoritmo di moltiplicazione: inizializzazione, iterazione, aggiornamento del conteggio e controllo della condizione di terminazione. L'automa di controllo (1ª versione): nei cerchietti solo i segnali asseriti. Osserviamo che lo stato ["NOADD,LOADA"] è superfluo, e si può eliminare, passando direttamente allo stato ["SHIFT"] se q_0=0. Questo comporta anche una variazione nella parte operativa (eliminazione del MPX); si tratta di un fatto generale: durante un progetto, cambiamenti nella parte operativa si ripercuotono sulla parte di controllo, e viceversa. Questa prima versione del controllo include il controllo della condizione di terminazione: dunque

non solo la PO, ma anche la PC dipendono dalla dimensione *n* degli operandi. Il tempo di calcolo è variabile da *n*+1 a 2*n*+1 colpi di clock, a seconda delle cifre a 1 nel moltiplicatore. 2ª versione del controllo: introduzione di una nuova condizione (FINE), che informa il controllo del raggiungimento della fine del conteggio. Si arricchisce la PO con un contatore down, al cui ingresso parallelo c'è *n*; la condizione FINE è il NOR delle uscite del contatore. Il conteggio viene aggiornato dalla PC attraverso il segnale SHIFT, che viene collegato all'ingresso di clock del contatore. Ora la PC è indipendente da *n*. Moltiplicazione con interi negativi in complemento a 2: si può usare la macchina appena vista, avendo cura di riportarsi al caso di moltiplicatore positivo. Versione alternativa del moltiplicatore: con registri di dimensione 2*n* (e più in generale $2n_{max}$) e uso di SHL e SHR. Esercizio macchine sequenziali: generazione di sequenze pseudocasuali con shift register.

I.11/30-32 – *giovedì 17 marzo 2005 (3 ore)*

Divisione di interi positivi: $D=dxq+r$. *D* è su 2*n* bit, gli altri su *n* bit. Divisione per 0. Overflow se $D-dx2^n \geq 0$. Il numero di bit di traboccamento in caso di overflow va da 1 a *n*. Algoritmo per la divisione: sottrazioni ripetute (*n+1* passi) - il primo passo valuta la condizione di overflow. Resti parziali. La sottrazione (SUB) si fa sempre, la memorizzazione del risultato solo se il segno non è negativo. Parte operativa: due registri da 2*n* bit (*A* e *M*, inizializzati rispettivamente a *D* e *d:0*), uno da *n+1* bit (*Q*, per overflow+quoziente). Scorrimento a destra per *M*, a sinistra (con scrittura seriale a destra del bit di quoziente calcolato ad ogni passo) per *Q*. Automa di controllo: ha una forma simile a quella del moltiplicatore (nella versione con ALU a 2*n* bit). Progetto di una macchina sequenziale per moltiplicazione e divisione di interi positivi: fusione delle parti operative, arricchimento del controllo (unione dei segnali necessari alle due macchine). Parte operativa: nasce un ingresso in più (=*MUL/DIV* che, insieme a NOP, deve essere fornito dall'esterno). Fusione degli automi di controllo: si eliminano tutti gli stati duplicati (stessa uscita). Se si fondono *N* macchine dedicate (ciascuna delle quali realizza una data operazione), l'automa complessivo ha *N* diramazioni dallo stato di attesa (*WAIT*). Dalle macchine dedicate al calcolatore: ciascuna operazione diventa un tipo di istruzione. Set di istruzioni: l'insieme delle operazioni elementari eseguibili dall'hardware di macchina. Opcode, operandi. Ad un alto livello di astrazione, l'automa di controllo di un calcolatore ha due stati: (1) fetch istruzioni, (2) esecuzione. Fetch (=prelievo) delle istruzioni: consente la lettura della prossima istruzione da eseguire. Nel fetch, l'istruzione viene prelevata dalla memoria, passando dal bus dati al registro *IR* (instruction register), che va in input all'unità di controllo, insieme alle condizioni dell'unità operativa.

I.12/33-35 – *martedì 22 marzo 2005 (3 ore)*

Tristate: terzo stato (= né 0, né 1), che consente di disconnettere elettricamente pur mantenendo la connessione fisica. Trasferimento dell'informazione all'interno della CPU: per i registri, si usano i comandi *Rin* e *Rout*. Per evitare un'eccessiva complessità interna, si introducono vincoli sul numero di unità che possono colloquiare simultaneamente tra loro - un caso estremo è offerto dalla strutturazione del trasferimento in un unico canale, o bus, che fa uso di tristate. CPU a singolo bus interno: ad ogni clock, solo un registro può porre un dato sul bus (*Rout*), mentre molti possono leggervi (*Rin*). Registri d'uso generale: $R_0 \ldots R_{n-1}$. PSW (=processor status word): contiene i flag di stato della parte operativa (es. overflow bit, carry bit, sign bit, etc.), che vanno in ingresso alla UC. Registri temporanei in ingresso e in uscita alla ALU. IR (=instruction register): contiene l'istruzione in esecuzione; è un altro degli ingressi alla UC. PC (=program counter): contiene l'indirizzo della prossima esecuzione da eseguire – tale indirizzo può essere modificato durante l'esecuzione dell'istruzione in IR (salti condizionati e non condizionati). MAR (=memory address register): ha le dimensioni del bus indirizzi. MDR (=memory data register): ha le dimensioni del bus dati. MAR e MDR sono i registri d'interfaccia con l'esterno per quanto riguarda indirizzi e dati – i controlli verso dispositivi esterni generati dalla UC viaggiano sull'omonimo bus. Altri ingressi alla UC: segnali provenienti dall'esterno del processore (es. MFC=memory function completed, dalla memoria attraverso il bus controlli) o da se stessa (es. END=ritorno al primo stato del fetch istruzioni). Gerarchia di programmi: di alto livello, assembly (linguaggio macchina),

microprogrammi. Dipendenza dalla macchina negli ultimi due casi. Compilatore e assemblatore. Ad ogni istruzione di un livello corrispondono più istruzioni al livello inferiore. Ad ogni istruzione assembly I_i corrisponde un microprogramma con un numero di microistruzioni N_i. Modi di indirizzamento: registro, diretto di memoria, immediato (l'unico caso in cui il dato risiede nella sezione di codice, all'interno dell'istruzione), indiretto (di registro e di memoria), base e indice (vettori), base, indice e scostamento (tabelle). "Effective address" dei dati in memoria.

I.13/36-37 – *mercoledì 23 marzo 2005 (2 ore)*

Microsequenze per le principali operazioni in CPU: (1) fetch istruzioni e incremento automatico PC, (2) trasferimento dati da e per la memoria (LOAD, STORE), (3) operazioni logico-aritmetiche (ADD, AND), (4) salti condizionati e non condizionati. Il PC va incrementato della lunghezza dell'ultima istruzione eseguita; tale lunghezza è variabile, in talune macchine (soprattutto CISC – ad es. l'8086 ha istruzioni lunghe da 1 a 6 byte). Per aumentare il parallelismo, si può usare un sommatore dedicato all'incremento del PC. Nei salti, si può codificare sia l'indirizzo assoluto della locazione di salto, sia l'offset; in entrambi i casi, questa informazione risiede in un opportuno campo di IR. Nel colloquio con memorie più lente del processore, vengono introdotti "cicli di wait": il segnale autogenerato WMFC pone la CPU in "folle", impedendole di cambiare stato fino all'arrivo di MFC dalla memoria; nel frattempo però la CPU può eseguire operazioni al suo interno. Convenzione per l'assemblatore della macchina a singolo bus interno: sorgente a sinistra, destinazione a destra. Esercizio proposto (compito dell'11/07/03): microprogramma per l'istruzione di macchina MUL RL_1,RL_2,R_3 – con accesso separato alla parte alta (H) e bassa (L) dei registri. Esercizio svolto (compito del 26/06/03): programma assembly per il calcolo della moltiplicazione RL_1*RL_2→ R_3, che fa uso delle istruzioni di macchina ADD, SHIFTL, SHIFTR, COMPARE, etc. (si ipotizza che la macchina non includa l'istruzione MUL nel suo set). Le istruzioni di salto condizionato fanno uso dei flag di stato (in PSW) generati dall'istruzione precedente. I due esercizi esemplificano rispettivamente l'approccio CISC e RISC al progetto di calcolatori.

I.14/38-40 – *giovedì 24 marzo 2005 (3 ore)*

Il tempo di esecuzione di un programma di alto livello è approssimabile con $T = N \times \underline{CPI} \times T_{clk}$, *con* $\underline{CPI}$ numero medio di cicli di clock per istruzione ed N numero di istruzioni di macchina effettivamente eseguite. La formula può servire per confrontare macchine diverse con stesso compilatore, o compilatori diversi per una data macchina. La frequenza di clock è solo uno dei parametri che condizionano le prestazioni. CISC: N basso, $\underline{CPI}$ alto; RISC: viceversa. Le prestazioni dipendono da tecnologia e architettura. Miglioramento architetturale delle prestazioni: legato alla possibilità di introdurre parallelismo nella macchina. Esempi: 3 bus interni, pipeline a quattro stadi: fetch F, decode D, Operate O, Store S (catena di montaggio, *cioccolatini Droste*). La durata di una singola istruzione non cambia, ma grazie al parallelismo, a regime ci sono quattro diverse istruzioni in esecuzione, e ad ogni passo ne viene completata una. Esercizio: programma assembly che calcola la somma di N interi consecutivi in memoria (vettore). L'indirizzo del vettore è quello del suo primo elemento. In memoria, il programma è suddiviso in due sezioni: codice e dati. Esempi di codifiche di macchina: (1) *move.b N,R2* prende 5 byte, (2) *move.b 0,R2* prende 3 byte, (3) *move.b R0,R2* prende 2 byte, nell'ipotesi che il bus indirizzi sia di 24 bit. Differenza tra *move num,*R_2 e *move #num,*R_2: nel primo caso si pone nel registro il contenuto della cella *num* (indirizzamento diretto), nel secondo l'indirizzo della cella, o *effective address* (indirizzamento immediato). Cicli di bus. Calcolo del numero di accessi al bus per l'istruzione *MOVE.DW R1,VAR*. Stack (it: pila) come memoria con regole: struttura dati di tipo LIFO (*esempio del lavapiatti e del cameriere)*. Le strutture FIFO si chiamano "code" – cfr. la pipeline. Un processo in memoria ha codice, dati, e stack. Si trasferiscono solo parole, non byte; lo stack cresce verso l'alto (indirizzi bassi). "Push" e "pop", stack pointer (SP), top e bottom of stack. Lo stack pointer punta all'ultima locazione scritta dello stack. L'instruction set dell'8086 prevede esplicitamente le istruzioni push e pop per i registri (es. PUSH AX) e per word di memoria (es. POP VAR[BX]). Uso dello stack:

procedure (o sottoprogrammi) e passaggio parametri (per valore o per indirizzo). Chiamata di e ritorno da procedura: le istruzioni 8086 call <nome_procedura> e return. La call è un jump con ritorno: nello stack si salva il PC.

I.15/41-43 – *martedì 5 aprile 2005 (3 ore)*

Esercizio: programma principale che chiama la procedura ACCUM che esegue la somma di N word del vettore NUM e scrive il risultato in SUM. ACCUM recupera i parametri (per valore: N, per indirizzo: NUM=#NUM[0], #SUM) usando l'istruzione MOVE con indirizzamento relativo di registro, che accede allo stack come ad una qualsiasi area di memoria RAM (R3 è inizializzato a SP, e poi modificato per puntare a parole differenti dello stack). Uso "trasparente" dei registri, ripristino della condizione iniziale dello stack mediante pop successivi o incremento dello SP. Memorie RAM (=ad accesso arbitrario). RAM statiche e dinamiche: elemento di memoria (nelle statiche è un FF, nelle dinamiche un microcondensatore), dimensioni tipiche (K/M), velocità (tempo di accesso ~1/~10 ns), consumo (H/L), costo per bit (H/L), organizzazione (per parola/per bit), uso tipico (cache/MM). Memorie dinamiche: refresh, address multiplexing, organizzazione per bit (per ridurre il numero di piedini dell'integrato). I principali piedini di un chip DRAM: indirizzi, dati, RAS, CAS, OE, CS, R/W, READY. Piedino CS: utilizzato per selezionare o meno il chip tra *K* chip posti sul bus ("espansione di indirizzo") – i CS sono tipicamente pilotati da un decoder. Piedino OE: abilita alla presentazione del dato sul bus. Durante il refresh si effettua una lettura speciale dalla memoria, in cui CS è asserito ma OE no. Il refresh non influisce molto sulle prestazioni. Controllore della memoria (genera i segnali necessari). Disco rigido: costo per bit, velocità, capacità. Coniugare capacità e velocità: gerarchie di memoria (un'altra idea architetturale): il processore "vede" una memoria grande quasi come il disco e veloce quasi quanto un registro. Sistema cache-MM: trasferimento di blocchi di dati, hit e miss, tassi di hit e miss (h+m=1). Come migliora il tempo di accesso: $T=hT_{cache}+mT_{MM}$. Una gerarchia tipica: cache di I e II livello, MM, disco. La gerarchia si basa sul principio di località (spaziale e temporale), che si applica sia a dati che a istruzioni. Esercizio: costruzione di un banco di memoria RAM D_{banco}=64M x 8 byte (parola dati d_{banco}=8 byte) con moduli (chip) da D_{chip}=16M byte e parola dati d_{chip}=2 byte. Il numero di moduli necessari (32) si ottiene dividendo D_{banco} per D_{chip}. I bit di indirizzo del banco (m_{banco}=26) vengono suddivisi in 2 gruppi: (1°) m_{chip}=23 vanno ai piedini di indirizzo di ciascun modulo (2°) m_{banco}-m_{chip}=3 vanno al decoder per il pilotaggio attraverso i CS di una delle 8 schiere da d_{banco}/d_{chip}=4 chip di cui è composto il banco (8x4=32).

I.16/44-45 – *mercoledì 6 aprile 2005 (2 ore)*

Nota sull'esercizio della lezione scorsa: una sua variante prevede $D_{banco}>D_{chip}$, ma $d_{banco}<d_{chip}$ (si veda l'esercizio d'esame del 27/06/2003); in questo caso, i bit di indirizzo del banco vanno divisi in 3 gruppi: (1°) m_{chip} ai piedini di indirizzo di ciascun modulo; (2°) $\log_2(D_{banco}/D_{chip})$ alla selezione del decoder per il pilotaggio del CS di uno tra D_{banco}/D_{chip} moduli; (3°) i restanti $\log_2(d_{chip}/d_{banco})$ alla selezione dei MUX/DEMUX sul bus dati. Diagramma temporale di una lettura da DRAM (diagramma "a caramella"). Protocolli di comunicazione tra moduli: sincroni, asincroni; questi ultimi non richiedono di conoscere la velocità dei dispositivi. Handshaking ("stretta di mano": asincrono). Cicli di wait. Cenni storici sull'8086/88 e sull'avvento del PC. Piedinatura (l'8088 è a 8 bit, l'8086 a 16): AD0-AD15, A16/S3-A19. Bus multiplexing: a seconda dell'istante di funzionamento, i piedini hanno un significato differente. ALE (address latch enable: simile a RAS e CAS, consente la memorizzazione esterna dell'indirizzo con bus multiplexato), BHE (=byte high enable, vedi seguito), MN/MX (modo minimo e massimo: nel modo massimo i segnali di controllo per l'esterno vengono prodotti da una macchina esterna all'808X), READY (il vecchio MFC), RESET, Vcc, GND. Modello di programmazione: 4 registri dati, o general purpose (AX,BX,CX,DX, ciascuno suddivisibile in una coppia di registri a 8 bit, es. AX=AH,AL), 4 puntatori o indici (SP,BP,SI,DI), 4 registri di segmento (CS,DS,SS,ES). Parola di stato PSW (processor status word): i flag CF (carry), PF (parity), ZF (zero), SF (sign), OF (overflow). La PSW include anche 3 bit di controllo: DF, TF, IF (cfr. seguito). Il 68000 ha il modo supervisore, impostabile in PSW; l'8086 no.

Segmentazione della memoria: l'indirizzo fisico PA (20 bit) si ottiene come combinazione di un registro di segmento (SR, 16 bit) e di un registro di scostamento (EA, "effective address" nella notazione INTEL, 16 bit) come PA=SRx16+EA (o simbolicamente: PA=SR:EA). I segmenti sono lunghi 64KB; la loro posizione in memoria è indicata da SR, mentre la posizione di un dato all'interno di un segmento è associata a EA. Segmenti consecutivi sono parzialmente sovrapposti. Stack Pointer = SS:SP, Program Counter = CS:IP (IP sta per instruction pointer). Vantaggi della segmentazione: suddivisione logica in codice, dati e stack; riduzione della frammentazione nella multiprogrammazione; agevole rilocabilità di un programma da parte del loader.

I.17/46-48 – *giovedì 7 aprile 2005 (3 ore)*

Rilocabilità: è il loader a decidere dove piazzare in memoria i vari segmenti; è quindi lui che provvede a inserire i valori di indirizzo reali nel codice. La segmentazione non evita al loader di dover modificare il programma eseguibile in presenza di riferimenti a indirizzi assoluti, ad es. MOV AX, SEG DATI. Operatori SEG e OFFSET - notare la differenza tra OFFSET Var (codifica: porzione di offset dell'indirizzo della variabile Var, dato indirizzato: porzione di offset dell'indirizzo della variabile Var, modo d'indirizzamento: immediato) e Var (codifica: porzione di offset dell'indirizzo della variabile Var, dato indirizzato: valore della variabile Var, modo d'indirizzamento: diretto). Struttura di un programma Assembly 8086: dichiarazione di segmenti, allocazione di memoria dati. Possono esserci più segmenti di codice, di dati o di stack in un programma, ma di volta in volta è attivo solo il segmento puntato dal relativo registro di segmento. Le direttive di allocazione di memoria DB e DW. L'operatore DUP. Inizializzazione di un dato: ? non inizializza. Calcolo dell'offset di un dato. Codifica delle istruzioni 8086: in mov AX, Var[SI] è implicito il registro di segmento DS (default=mancanza di indicazione contraria). Ancora sulle memorie: ordinamento e allineamento. Ordinamento: byte-addressable computers, little endian (8086) vs big endian (68000). Allineamento nell'8086: lettura di byte/parole ad indirizzi pari/dispari. Due banchi da 512K, gestiti tramite BHE e AD0. In sostanza, si usano 21 bit anziché 20 per l'indirizzamento di 1Mbyte. La lettura di parole ad indirizzi dispari impiega due cicli di bus. Struttura interna dell'8086: EU (execution unit)/BIU (bus interface unit), a realizzare una primitiva pipeline. Coda di prefetch (6 byte), e suo riempimento evitando letture di parole ad indirizzi dispari. Il set di istruzioni di macchina (cfr. sito web). Le tipologie di istruzioni sono (1) data transfer (es. MOV, PUSH), (2) arithmetic (es. ADD, CMP), (3) logic (es. OR, TEST), (4) string manipulation, (5) control transfer (es. CALL, JMP, JLE/JNG), (6) processor control. Le istruzioni CMP e TEST sono equivalenti a SUB e AND, ma non "sporcano" i registri, modificando solo la PSW. Modi di indirizzamento dati con 8086: immediato (71), registro (BH), diretto (Var), indiretto di registro ([BX]), relativo con registro indice (Var[SI]), indiretto con registri base e indice ([BX][SI]), relativo con registri base e indice (Var[BX][SI]). Esercizio: trovare l'indirizzo fisico del dato Var[BX][SI]; soluzione: PA=DS:EA, con EA=Offset VAR+BX+SI. Codifica in linguaggio macchina delle istruzioni a due operandi: tabella dei modi di indirizzamento (cfr. sito web). Codifica completa dell'istruzione ADD AL,Var[SI]: si parte dal manuale, che dà per ADD reg/reg o mem/reg la codifica a 4 byte 000000DW MOD|REG|R/M (DISP-LO) (DISP-HI); nel nostro caso si ottiene (D=1 perché reg è destinazione, W=0 perché trasferimento di byte, inoltre sia offset Vect=2345H) con l'aiuto della tabella dei modi di indirizzamento 00000010 10|000|100 D16-lo D16-hi = 02 84 45 23 (hex). VECT è uno spiazzamento a 16 bit (=D16); col modo di indirizzamento [BX+2] invece si codifica il 2 con un D8. Il registro di segmento implicito è DS. A una singola istruzione assembly possono corrispondere più codifiche in linguaggio macchina (es. SUB AX,BX si può codificare con 2BC3 – AX è destinazione - o con 29D8 – BX è sorgente). Ci sono codici macchina abbreviati per istruzioni – utilizzate di frequente - che lavorano con registri particolari (es. "add immediate to register" richiede 4 byte, mentre "add immediate to accumulator (=AX o AL)" ne richiede solo 3).

I.18/49-51 – *venerdì 8 aprile 2005 (3 ore)*

Assemblaggio (masm) e collegamento (link) di moduli assembly. Il caricamento in memoria è fatto dal sistema operativo (loader). Il masm non fa distinzione tra maiuscole e minuscole (ossia è case insensitive). Analisi dettagliata del programma *main*. Il file main.lst ha in più la numerazione delle righe e la codifica di dati e istruzioni. *main* è composto da due moduli: main.asm e subrout.asm. Moduli e segmenti: spezzoni di uno stesso segmento possono essere definiti in moduli diversi: sarà il linker a combinare gli spezzoni in un unico segmento. Dichiarazione di segmenti: direttive di allineamento (es. PARA: allineamento al paragrafo, ossia ad un indirizzo multiplo di 16), combinabilità (es. PUBLIC: pone gli spezzoni di segmento di uguale nome in aree contigue di memoria), classe (es. 'STACK': pone i segmenti della stessa classe in aree contigue di memoria). Dichiarazione di costanti (EQU): serve solo durante l'assemblaggio. Differenza tra direttive e istruzioni. Commenti: una riga (;) oppure più righe (COMMENT). Codice: la direttiva ASSUME: *non* comporta il caricamento automatico dei registri DS ed ES, che rimane compito del programmatore. Ritorno al DOS come ritorno da una procedura. Metodo alternativo: uso della trap INT 21h, funzione 4Ch (la funzione richiesta va messa in AH prima della chiamata alla trap). Traps=servizi messi a disposizione dal sistema operativo. La trap INT 21H (funzione 09h: stampa di stringa terminante con '$'). Allocazione di memoria per i dati (DB, DW) e codifica delle istruzioni. L'operatore DUP. Inizializzazione di un dato: ? non inizializza. Location counter: conta i byte allocati in ciascun segmento. Il valore del location counter viene ritoccato dal linker quando combina spezzoni di segmento in un unico segmento. Rilocabilità: è il loader a decidere dove piazzare in memoria i vari segmenti; è quindi lui che provvede a inserire i valori di indirizzo reali nel codice (si veda la codifica parziale dell'assemblatore "---- R"). Comunicazione tra moduli: EXTRN (nel modulo dove la procedura è usata), PUBLIC (nel modulo dove la procedura è definita). Etichette: di tipo NEAR o FAR (codice), BYTE o WORD (dati). Dichiarazione di etichette: forma implicita (es. Pippo:) ed esplicita (es. Pippo Label near). Dichiarazione di procedure. Macro: sua dichiarazione, e sue differenze dalla subroutine (si risparmia in tempo di esecuzione evitando la chiamata a procedura, ma si perde in spazio occupato). Operatori SEG e OFFSET - notare la differenza tra OFFSET Var (codifica: porzione di offset dell'indirizzo della variabile Var, dato indirizzato: porzione di offset dell'indirizzo della variabile Var, modo d'indirizzamento: immediato) e Var (codifica: porzione di offset dell'indirizzo della variabile Var, dato indirizzato: valore della variabile Var, modo d'indirizzamento: diretto). Assemblaggio/collegamento ed esecuzione del programma *main*. Errori in fase di assemblaggio (es. errori di sintassi: "MOVE" anziché "MOV", omissione del carattere ";" prima di un commento, etc.), collegamento (es. definizione in main.asm di una procedura esterna che non viene poi trovata in altri moduli dal linker), esecuzione (es. omissione del carattere "$" nella stringa SALVE: verrà stampato il contenuto del segmento dati DATA fino al "$" della stringa CIAO). Warning e severe errors. "Warning: no stack segment" è generato dal clinker, e non dall'assemblatore.

I.19/52-54 – *martedì 12 aprile 2005 (3 ore)*

Interfacce di I/O: consentono la comunicazione tra CPU e periferici – cfr. libro Bucci. Interfacce di ingresso, di uscita, di ingresso/uscita, parallela, seriale. Registri caratteristici: dati (in sola lettura nelle interfacce d'ingresso, in lettura/scrittura nelle comunicazioni bidirezionali, in sola scrittura nelle interfacce d'uscita), di stato (di sola lettura), di controllo (di sola scrittura). Decodifica degli indirizzi di porta. Due registri possono essere mappati su una sola porta, se uno è di sola lettura e l'altro di sola scrittura: sarà il segnale R/W a discriminare. Indirizzamento di I/O con spazio di indirizzamento separato: sono necessarie istruzioni diverse per trasferimenti con I/O e con memoria, cui corrisponde un segnale di controllo di tipo M/IO. L'8086 usa le istruzioni dedicate IN e OUT. Esempio di interfaccia di uscita (stampante), ad una sola porta, per 8088: parte operativa (l'hardware di interfaccia), parte di controllo (il software di gestione in assembly 8088). Metodi per la gestione dell'I/O: (1) a controllo di programma (o polling, cfr. esempio precedente), (2) sotto controllo di interruzione, (3) tramite DMA – Direct memory access. Interruzioni, o interrupt: migliorano l'efficienza dei trasferimenti rispetto al controllo di programma, costringendo i periferici a richiedere l'attenzione – ossia le risorse macchina - della CPU. La CPU serve un dato dispositivo

facendo saltare il controllo alla relativa interrupt service routine. Differenza tra routine di interruzione e subroutine: le prime devono essere assolutamente “trasparenti”. Linea di INTR (interrupt request) unica: la CPU deve risolvere l’identità e la priorità dei dispositivi che hanno fatto richiesta. Due principali soluzioni: (1) polled interrupt, che controlla secondo una sequenza prefissata i bit di stato delle interfacce di I/O – con i problemi di efficienza tipici del polling, (2) vectored interrupt, in cui su richiesta della CPU (INTA, interrupt acknowledgement) un codice (interrupt type) che identifica il dispositivo a massima priorità viene posto sul bus dati, indi letto dalla CPU e internamente convertito nel vettore di interruzione (indirizzo della routine di servizio) da porre nel PC prima della conclusione dell’istruzione corrente. L’int type non è un indirizzo, bensì rimanda ad un indirizzo – questo meccanismo è utile per due motivi (minore informazione da trasmettere sul bus dati, maggiore flessibilità di gestione da parte della CPU). Passaggio dal tipo al vettore nell’8086: la tabella delle interruzioni (che risiede all’indirizzo 0, e contiene 256 vettori). Il vettore di interruzione ha 32 bit (il salto è sempre di tipo intersegmento, o FAR).

I.20/55-57 – *martedì 12 aprile 2005 (3 ore: esercitazione di laboratorio facoltativa)*

Assemblaggio/collegamento ed esecuzione del programma *main*. Errori in fase di assemblaggio (es. errori di sintassi: “MOVE” anziché “MOV”, etc.), collegamento (es. definizione in main.asm di una procedura esterna che non viene poi trovata in altri moduli dal linker), esecuzione (es. omissione del carattere “$” nella stringa SALVE: verrà stampato il contenuto del segmento dati DATA fino al “$” della stringa CIAO). Debug (con dbx main.exe): dump della memoria, verifica di cambiamenti nel contenuto dei registri. Il pop-up menu LW86. Disassemblatore: dal codice macchina al codice ASM (a meno delle etichette definite dal programmatore, sostituite da costanti numeriche – es. 0D per CR - e indirizzi – es. 001C per CIAO). A un singolo indirizzo fisico possono corrispondere più coppie SR:EA (es.136C:0140=1380:0000=13800). Segment override prefix: consente di specificare registri di segmento alternativi al default (es PUSH ES:parola; la codifica di questa istruzione richiede un byte in più rispetto a quella col default). Esercizi sulla manipolazione di stringhe: il programma scara.asm (una versione commentata del codice ASM 86 scritto è disponibile in rete).

I.21/58-60 – *mercoledì 13 aprile 2005 (2 ore)*

Il vectored interrupt I/O utilizza un particolare cablaggio dei dispositivi (daisy chain, semplice ma poco flessibile) o un controllore ad hoc (PIC, programmable interrupt controller), attraverso il quale è possibile programmare la priorità. Cosa si intende per simultaneità nell’arrivo di una richiesta di interruzione. Classificazione delle interruzioni: sincrone/asincrone, prevedibili/non prevedibili (cfr. libro Bucci). Accettazione di un’interruzione esterna con 8086: prima di passare il controllo alla routine di interruzione, la UC salva nello stack CS, IP e PSW, lasciando alla routine di interruzione il compito di salvare i registri che usa. Ritorno dall’interruzione: IRET (differisce dalla RET perché è sempre FAR e ripristina anche la PSW). La UC pone a zero TF (trace: utilizzato ad esempio per il debugging step-by-step) e IF (mascheramento delle interruzioni esterne). L’istruzione STI all’interno di una routine di servizio ne autorizza l’interrompibilità. Gli interrupt si dividono in hardware e software (detti anche “traps”; nell’8086 si chiamano con INT <type>), mascherabili e non (nell’8086 vanno sul piedino NMI; es. power failure, clock), interni (“eccezioni”) ed esterni. Gli interrupt interni e quelli software non sono mascherabili. Tabella delle interruzioni 8086 (INTEL, BIOS, DOS): divide by 0, trace, clock, keyboard interrupts. Rivettorizzazione (o ridirezione, o intercettamento) di un’interruzione: comporta l’inserimento di una nuova coppia di valori CS:IP nella tabella delle interruzioni (esempi: rivettorizzazione del tipo 1CH – dummy return – con un programma che mostri l’ora corrente, ridirezione del tipo 09 – tastiera – per la realizzazione di un nuovo driver di tastiera). Modifiche all’interfaccia per stampante a controllo di programma, con inserimento della possibilità di controllo sotto interruzione: si aggiunge un FF di controllo, che la CPU può porre a 1 per abilitare le interruzioni di un dato dispositivo, o per mascherarle in modo selettivo – le porte diventano due (cfr. libro Bucci). Controllo sotto interruzione: il driver di stampa è composto da due sezioni: inizializzazione (eseguita una sola volta all’inizio del trasferimento) e

stampa di un singolo carattere (è la routine di servizio dell'interruzione). La sezione di interruzione fa uso di variabili globali (BUSY, IND, CONT, inizializzate dalla sezione precedente), che tengono traccia dello svolgimento del task di stampa. Accesso diretto alla memoria: il controllore DMA può svolgere il ruolo di bus master, facendo richiesta (bus request) alla CPU, che risponde con un bus grant. Bus busy (durante il trasferimento DMA), bus release (rilascio del bus alla CPU). Il DMA si usa per evitare che i dati passino per i registri di CPU. Durante i trasferimenti DMA, la CPU è libera di svolgere calcoli che non richiedano il bus. Il DMA viene inizializzato da programma, specificando l'indirizzo del trasferimento e la dimensione del blocco dati da trasferire. Il trasferimento per blocchi è tipico dei dischi, che hanno elevato tempo di latenza ma anche elevato bit rate. DMA usato anche per stampanti veloci.

I.22/61-63 – *giovedì 14 aprile 2005 (3 ore: esercitazione di laboratorio facoltativa)*

Algoritmo "3x+1": il programma 3xp1.asm (una versione commentata del codice ASM 86 scritto è disponibile in rete). Input da tastiera, conversioni di formato (da intero a stringa e viceversa).

I.23/64-66 – *venerdì 15 aprile 2005 (3 ore: esercitazione facoltativa)*

Esercizi di preparazione al compito. *Rappresentazione*: conversione (in due modi diversi: divisioni successive per 8, o passando dalla rappresentazione base 2) di un numero decimale in formato ottale. *Reti combinatorie*: priority encoder a quattro ingressi (due versioni: monoblocco, e come cascata di due blocchi, il primo codificatore di priorità su quattro uscite, il secondo un semplice decoder binario 4/2). *Reti sequenziali*: analisi di un circuito con 2 FF JK. Equazioni caratteristiche, tabella delle transizioni, diagramma degli stati. Disegno dell'uscita dato un ingresso variabile nel tempo. Il circuito funziona da riconoscitore della sequenza di ingresso 101. *CPU*: codifica (plausibile) dell'istruzione PUSH.DW [PVAR], e calcolo del numero di cicli di bus necessari al suo fetch+esecuzione. L'operando è specificato secondo il modo di indirizzamento indiretto di memoria, non presente nell'8086. Il push è di una double word. Il bus di indirizzi è a 24 bit, il bus dati è a 16 bit. *Memoria*: si veda la soluzione simbolica al problema della costruzione di un banco di memoria riportata più sopra (lezioni del 5 e 6 aprile). *Hardware 8086*: accesso alla tabella delle interruzioni. I valori di salto per IP e CS sono ivi memorizzati in forma little-endian. *I/O* e *Software 8088*: progettazione di un'interfaccia che serva tre dispositivi di input (8 bit), e suo controllo (modalità polling) attraverso una procedura assembly. Quattro porte: tre per i dati, una per lo stato (2 bit). La porta di stato è una pseudo-porta, poiché non ha FF, ma solo un buffer tristate. Generazione della priorità, convertita poi in parola di stato. Il software deve verificare innanzitutto la presenza di un dato da leggere, e poi stabilire l'identità del dispositivo che invia il dato. Il software deve memorizzare separatamente i dati immessi dai tre dispositivi, tenendo traccia anche di quanti dati sono stati letti da ciascun dispositivo.