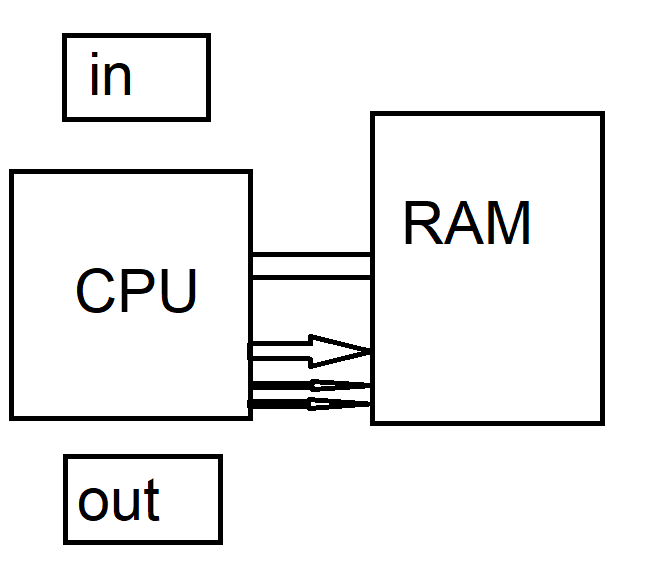
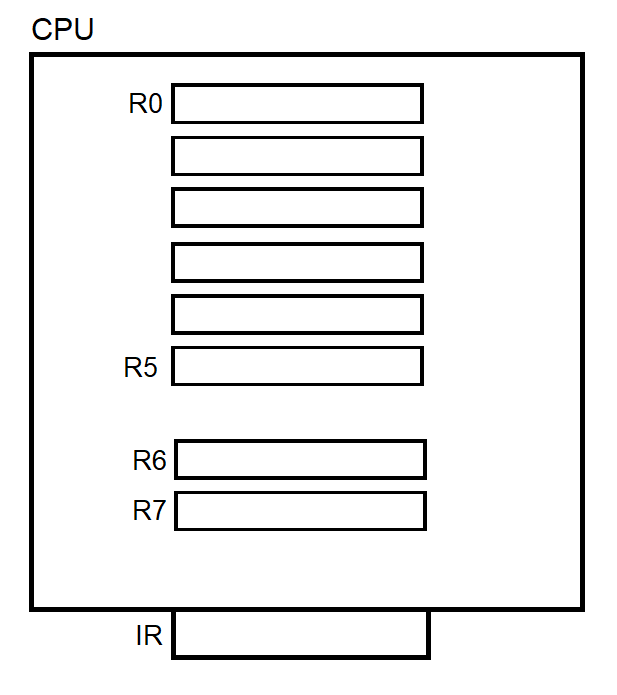
Abbiamo visto un esempio di calcolatore primordiale: la macchina di Von Neumann



Ora vedremo un esempio di architettura di CPU che oramai non è più in uso (se non per progetti aerospaziali wtf?). Essa fu sviluppata negli anni ’50 (e fu anche l’architettura che ci portò sulla Luna WOW), ha il pregio di essere abbastanza semplice da spiegare.



Ci sono 6 Registri di uso generale (R0, R1, R2… R5) e 2 che ricoprono ruolo particolare, il registro R6 (SP) e il registro R7 (PC). I registri sono organizzati su celle da 16 bit. Anche le celle della RAM sono da 16 bit e per l’indirizzamento vengono usati 15 fili (ci sono al massimo 32768 celle di memoria, per un totale di 64 Kb).

Il registro delle istruzioni è l’unico che non può essere manipolato dalle istruzioni (infatti ha una posizione particolare).

La RAM può ospitare però sia dati di tipo byte (quindi che occupano solo 8 bit), che word (16 bit), che double word (32 bit).

Per la codifica delle istruzioni si utilizzano 4 bit per il codice operativo, e poi ci sono 2 espansioni possibili, quella a 7 bit e quella a 10 bit. Le due combinazioni di espansione sono quella con -111 (nella seconda, terza e quarta cifra binaria) per i 7 bit e con -000 (sempre nel 2°, 3° e 4° bit) per i 10 bit. Il bit più significativo (il primo) indica se si sta lavorando con un byte o con un word (1 indica byte, 0 indica word).

Nel caso del codice a 4 bit, ne restano 12 per rappresentare gli operandi, 6 dedicati al sorgente e 6 alla destinazione. Questi 6 bit sono rappresentati come 3 + 3 (3 indicano il modo di indirizzamento, gli altri 3 individuano uno degli 8 registri del processore, sono quindi compresi quelli speciali -> il Program Counter/R7 è manipolabile). I tipi di indirizzamento sono:

000 registro

001 indiretto mediante registro

010 autoincremento

011 autoincremento indiretto

100 autodecremento

101 autodecremento indiretto

110 indicizzato (indexed)

111 indicizzato indiretto.

Per Esempio:

000011 vuol dire “prendi come sorgente il CONTENUTO del registro 3”

001010 vuol dire “prendi come sorgente RAM[R2]”

010101 vuol dire “prendi come sorgente RAM[R5] e poi: R5 <- R5 + 1 se si opera su singolo byte; R5 <- R5+2 se si opera sul tipo word a 16 bit”.

011101 vuol dire “prendi come sorgente RAM[RAM[R5]] e poi:” <incremento di R5 come sopra> (questo permette di inserire in RAM una “tabella” di indirizzo / un array di puntatori)

100101 fa l’opposto dell’autoincremento: “R5 <- R5 -1 e poi: prendi come sorgente RAM[R5]”

101101 vuol dire “R5 <- R5 -1 e poi: prendi come sorgente RAM[RAM[R5]]” (è come l’autoincremento indiretto ma fa l’autodecremento. Nota: le somme e sottrazioni di 1 potrebbero essere di 2 a seconda del tipo di dato con cui si lavora).

110000 vuol dire “prendi come sorgente RAM[R0 + costante]”. La costante deve essere da 16 bit, quindi l’istruzione viene codificata in una “parola” di memoria e se presenta indirizzamento indicizzato al valore contenuto nel registro viene sommato il valore contenuto nel registro successivo (quindi in quel caso è come se l’istruzione fosse codificata su una parola in più).

111000 vuol dire “fai l’indirizzamento indicizzato ma per raggiungere una cella di memoria RAM che in realtà contiene l’indirizzo del valore da salvare come sorgente, in formula RAM[RAM[R0 + costante]]”

Quando andiamo a prendere come sorgente il PC possono succedere delle cose interessanti, soprattutto se prendiamo come indirizzamento il modo di autoincremento. In questo caso, si ha appena fatto il fetch di un’istruzione, quindi il PC è già incrementato di uno, e nella cella a cui adesso sta puntando sarà presente il valore da salvare nel registro sorgente, ma per la proprietà dell’autoincremento dopo aver fatto ciò il valore nel PC aumenta nuovamente di 1, così che adesso invece punti alla istruzione da eseguire successivamente (è un caso di indirizzamento immediato o diretto). Analogamente avviene quando si usa l’autoincremento indiretto sul PC, anche se in quel caso l’indirizzamento assume la forma di un indirizzamento indiretto mediante RAM.

Tra le istruzioni più semplici vi è la MOVE: essa viene codificata con x001 (con il primo bit che è 0 se l’operando è una word, 1 se è un byte). Quel che fa è prendere il contenuto nella sorgente e lo invia nella destinazione. Può tornare utile per muovere un valore dalla RAM a un registro interno della CPU.

Per esempio l’istruzione:

0001 011111 000101 (16 bit)

Carica nel registro R5 il contenuto della cella di memoria RAM il valore contenuto nella cella puntata dalla cella RAM successiva (quella a cui sta puntando il PC, che è R7).

Mentre invece

0001 000100 011111 (16 bit)

Carica nella cella di memora RAM puntata dalla cella RAM successiva (quella puntata dal PC) il valore contenuto nel registro R4.

Invece

0001 000100 000011

Carica nel registro R3 il valore contenuto in R4.

Queste erano tutte istruzioni abbastanza semplici, ma la parte complicata arriva adesso. Consideriamo per esempio l’istruzione di CONFRONTO (COMPARE): essa permette di effettuare un calcolo, che consiste nella differenza tra l’operando Sorgente e l’operando Destinazione. La cosa interessante di questa istruzione è che il risultato calcolato in realtà viene buttato via: il motivo per cui si fa questa differenza è per andare a verificare le condizioni che la ALU fornisce sulle uscite ausiliarie: tali informazioni vengono salvate in un registro particolare, il “Registro di Stato”. I bit del registro di stato memorizzano se il risultato è stato negativo / se l’operazione è andata in overflow / se il risultato è 0 ecc. (il risultato = 0 corrisponderebbe alla verità dell’uguaglianza)

Questi dati servono perché rimanendo salvati in un registro possono essere utilizzati dalle istruzioni successive per fare dei “salti” di tipo condizionale. Nel caso dell’istruzione di COMPARE, quindi, il vero registro di destinazione è il registro di stato (quello indicato come dest nella “word” di istruzione è in realtà come se fosse sorgente).

Un’altra istruzione che va a modificare il registro di stato è l’operazione BIT SET (l’AND logico)

Ricapitolando:

MOVE = X001

COMPARE = X010

BIT SET = X011

Ce ne sono anche altre: L’OR (= BIT TEST), Il Cambio di Segno (BIT CLEAR, in complemento a 1?) ecc.

Tra queste, importante è l’operazione di somma e quella di sottrazione:  
ADD = 0110  
SUB = 1110

Interessante è che entrambe queste operazioni sono svolte su 16 bit (quindi in realtà il bit che normalmente viene usato per distinguere tra byte and word va a farsi f… e viene usato per distinguere tra i due tipi di funzione). Inoltre, queste due operazioni usano il registro di destinazione sia come destinazione che come operando (quindi il suo valore viene perso e sostituito dal risultato) -> Dest = Dest + Source o Dest = Dest – Source.

DA NOTARE: la sottrazione, a differenza dell’addizione, NON è commutativa, quindi bisogna tenere bene in conto che il valore Dest sarà quello a sinistra del segno -.

Nel caso dell’espansione a 7 bit il 2°, il 3° e il 4° hanno il valore costante 1, quindi i bit che rimangono disponibili per la codifica sono il primo (che svolge il solito ruolo) e gli ultimi 3 (quindi si aggiungono altre 16 istruzioni). Dei restanti 9 bit, 3 vengono usati per identificare un registro interno (quindi il metodo di indirizzamento è costante) e gli altri 6 vengono usati per identificare il secondo operando al modo solito (3+3 -> modo di indirizzamento + indirizzo).

Un esempio di questo tipo è l’istruzione XOR: Uno dei due operandi deve per forza essere contenuto negli 8 (in realtà 6) registri (di uso generale), l’altro può essere dove gli pare. Anche in questo caso, come nella somma e nella sottrazione, il secondo operando viene sovrascritto. Il codice operativo dello XOR è 01110100.

L’operazione di Moltiplicazione prende come ingresso operandi su una Word e produce risultati su 2 Word (questo per evitare facili overflow). Anch’esso ha 7 bit di istruzione, quindi riceve i valori sorgente come lo XOR, però salva il prodotto della moltiplicazione nel registro indicato come Sorgente1 (quindi quello interno di uso generale) e nel suo Successivo. Esempio:

0111000 010 010111

Moltiplicazione: Moltiplica il contenuto di R2 per il valore della cella RAM a cui punta il PC (R7) e salva il risultato in R2R3 (i bit più significativi vanno in R3).

Le coppie utilizzabili sono R0R1, R1R2, R2R3, R3R4, R4R5 (quindi R5 non può essere Sorg1 poiché R5R6 non è utilizzabile).

Un altro tipo di operazione interessante è quella di scorrimento aritmetico. L’istruzione ha codice operativo 0111010. La cosa interessante è che il registro è operando Sorgente e Destinazione. Il valore contenuto nei bit che normalmente indicherebbero la Destinazione, quindi, indica di quanto scorrere il valore (è rappresentato in complemento a 2, quindi i valori di quei 6 bit vanno da -32 a +31, dove ogni valore indica un bit). Mentre lo scorrimento a destra cancella dei bit, lo scorrimento a sinistra può andare a causare un overflow (quindi può modificare il registro di stato)

Il codice operativo 0111011 (o 1111010 non ho capito) effettua lo SCORRIMENTO COMBINATO (esso si effettua su una coppia di registri -> se specificassimo R2 come Sorgente andrebbe a salvare il valore in R2R3 -> ciò spiega anche la necessità del range di scorrimento a 32).