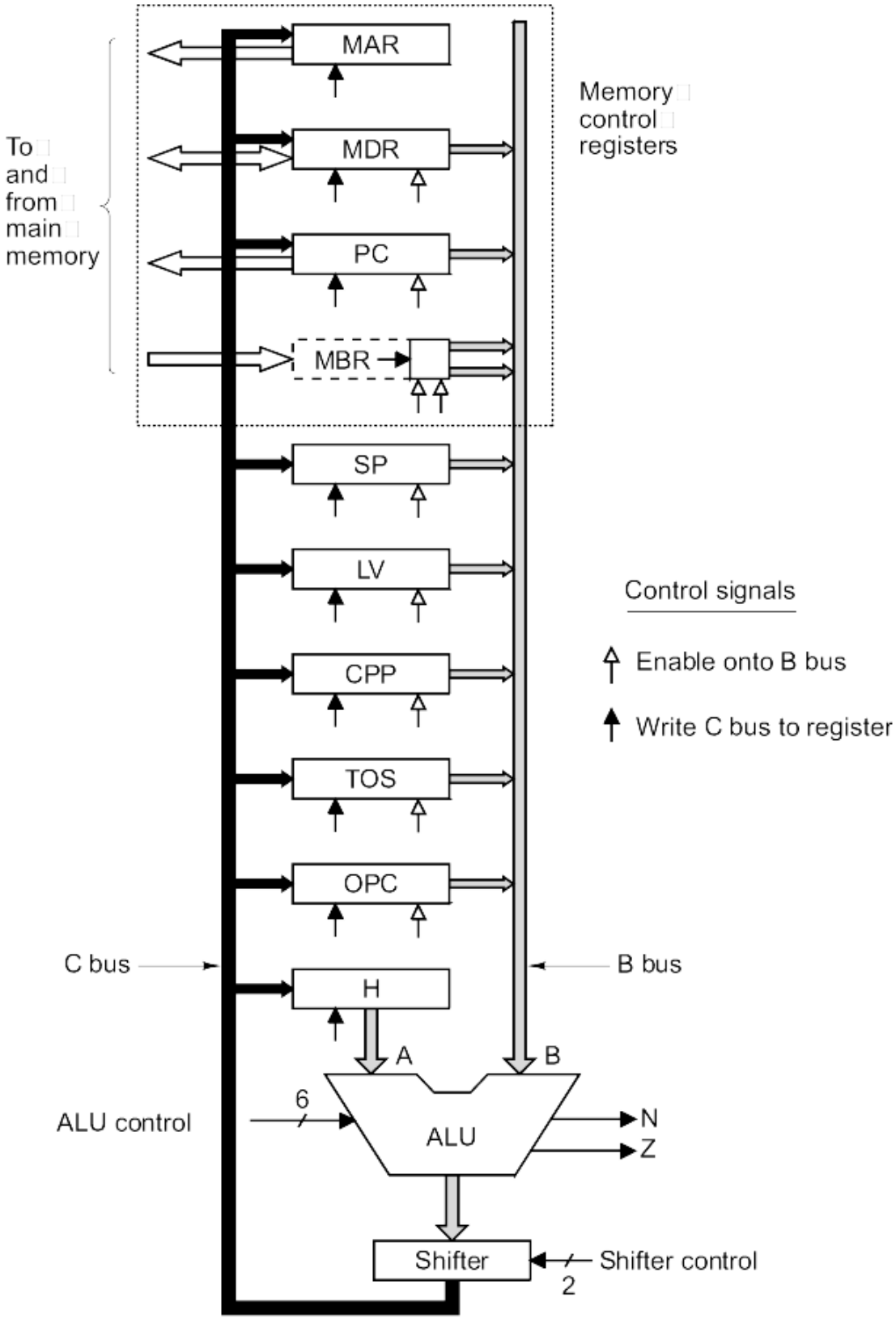
Microarchitettura

Il linguaggio **MAL (Micro Assembly Language)** dispone di alcune specifiche come: un numero **limitato di costanti**(0, 1 e -1), un **numero limitato di variabili** che sono i registri. Dobbiamo scegliere dov'è **memorizzato il dato** e come allocarlo.

Le **operazioni aritmetiche sono limitate** e in oltre anche i **salti condizionali** devono essere **collegati da label**.


MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
To and from main memory
Memory control registers
Control signals
Enable onto B bus
Write C bus to register
C bus
B bus
A
B
6
ALU control
ALU
N
Z
Shifter
Shifter control
2

Ogni istruzione ISA viene chiamata dalla memoria, presa dalla una sequenza di istruzioni e scelta in base a dei criteri. La variabile che sceglie l'istruzione è il PC (Program Counter) il quale punterà poi alla prossima istruzione.

Il percorso dei dati inizia con **MAR**, **MDR** e **PC** inviano dati al **Bus B,** il quale è un input della ALU. La ALU è formata da 6 linee di controllo: **ENA**, **ENB**, **INVA**, **INC**, **input A** e **input B**.

Ci sono due registri **N** e **Z** che servono per i salti condizionali. Ci sono due tipi di shift: **SLL8** e **SRA1** il primo esegue uno spostamento verso sinistra di 1 byte e imposta gli altri a 0. Il secondo invece sposta tutto a destra di 1 bit lasciando invariato il numero più significativo che è il segno.

Per incrementare il PC si porta il suo valore sul Bus B, successivamente si disabilita il Bus A (entrata sinistra ALU) e si abilita il segnale di INC, si memorizza il nuovo risultato di PC.

Un altro registro importante è l'**H** ovvero holding.

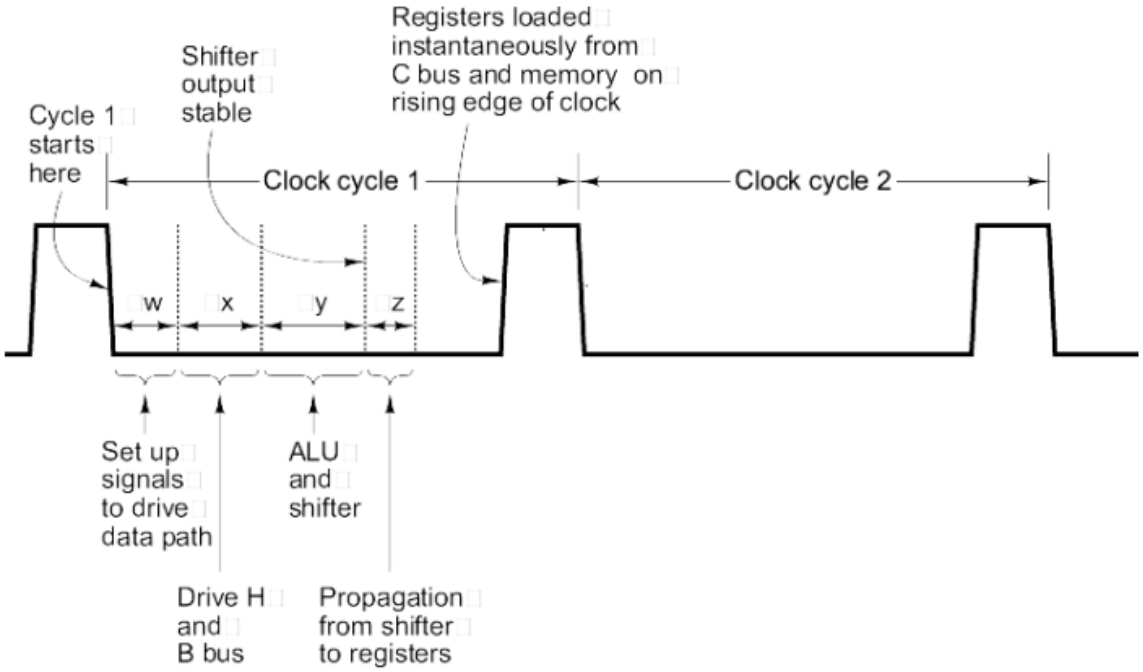
Ogni istruzione è formata da **opcode** e **operando.**

Sincronizzazione del Datapath

Quando inizia un nuovo ciclo di clock, viene generato un nuovo impulso. Questo impulso si propaga per un breve periodo che può essere diviso in quattro sezioni distinte:

1. Δw è il fronte di discesa, nel quale vengono impostati i segnali delle porte logiche, questi segnali sono presi dai registri della CPU.
2. Δx si imposta H e si stabilizza il Bus B.
3. Δy al suo inizio la ALU e lo shifter lavorano sui dati e al suo termine si stabilizza il risultato.
4. Δz il risultato passa al Bus C.

Al fronte di risalita i risultati vengono caricati sui registri e così si salvano per l'inizio del prossimo ciclo di clock.



Ci sono due momenti distinti durante il periodo di clock: **discesa** dell'impulso che permette **l'inizio** del percorso dei dati e la **risalita** dell'impulso che permette la **memorizzazione** dei dati nei registri.

Interazione con la memoria

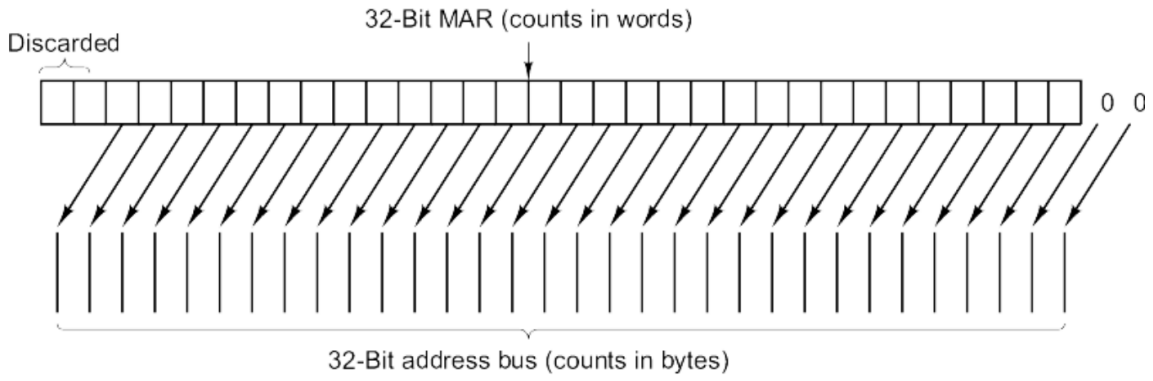
Ci sono due metodi per comunicare con la memoria dalla CPU:

1. Una porta a 32 bit per lettura/scrittura dei dati a livello ISA, controllata da due registri:
    a. **MAR** (Memory Address Register) specifica l'indirizzo di memoria sul quale scrivere o leggere una parola.
    b. **MDR** (Memory Data Register) ospita la parola che sarà letta o scritta in base al **MAR.**
2. Una porta a 8 bit per lettura, fetch ISA controllata da due registri:
    a. **PC** (Program Counter) il quale indica l'indirizzo della **prossima istruzione ISA** da caricare.

b. **MBR** (Memory Byte Register) contiene il byte letto dalla memoria durante il **fetch**. Essendo un registro a 32 bit, il contenuto è caricato negli 8 bit meno significativi.

Tutti i segnali sono controllati da **uno** o **due** segnali di controllo. Nella figura del data path le frecce vuote sono i segnali di output sul Bus B mentre quelle piene sono di caricamento dal Bus C.

Essendo che MAR/MDR e PC/MBR usano rispettivamente per parlare con la memoria 32 bit (word) e 8 bit (byte) abbiamo due situazioni diverse. Può sembrare un'apparente complicazione ma in realtà semplifica la **fetch** di una istruzione che avviene un **byte per volta,** mentre **leggere/scrivere** una **parola** (32 bit) in un **unico ciclo**.

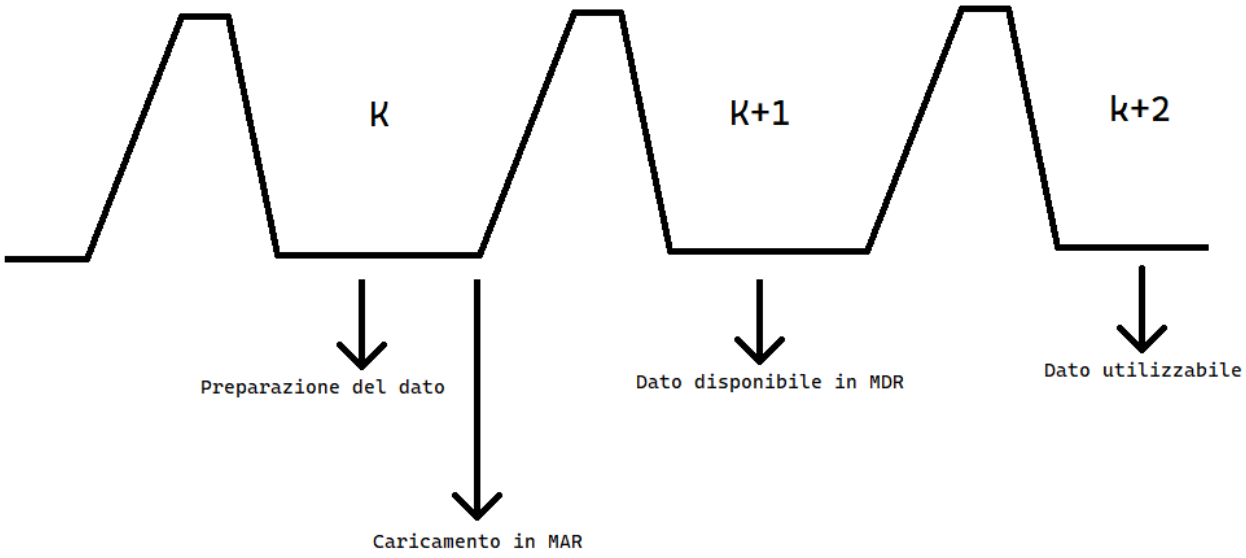


Il trucco per far funzionare questo meccanismo senza aggiungere circuiti è di sfalsare di 2 bit il collegamento, facendo rimanere inutilizzati i 2 bit più significativi della **MAR.** E mettendo a 0 i meno significativi. Così facendo il bit "zero" della MAR sarà collegato con il bit 2 dei bus, il bit "uno" della MAR sarà collegato al bit 3 dei bus e così via.

Essendo che MAR può indirizzare (come un puntatore) delle **word (32 bit)**, i primi due bit sono stati posti a 1 così che quando il primo bit è 1 (100) il valore è 4, tutti i successivi sono multipli di 4 e quindi 8, 12, 16 e così via per poter esprimere i valori senza doverli indirizzare in altri modi, ovviamente questo a discapito degli ultimi due bit che si vanno a perdere.

In conclusione MAR e MDR sono rispettivamente un puntatore e un buffer che grazie alla CPU leggono o scrivono dati dalla memoria mentre PC e MBR servono per leggere dati e solo questo possono fare.

Non è possibile completare un ciclo di scrittura o lettura in una volta sola, ci vogliono più cicli per completare l'operazione.



I dati quindi saranno utilizzabili due cicli dopo la richiesta (k+2) di questi ultimi, ciò non obbliga la CPU nel ciclo k+1 a rimanere inutilizzata, infatti in quest'ultimo può essere eseguito un **altro ciclo** di data path che non necessita del dato richiesto.

Microistruzioni

Nel data path sono presenti 29 segnali di controllo, di cui:

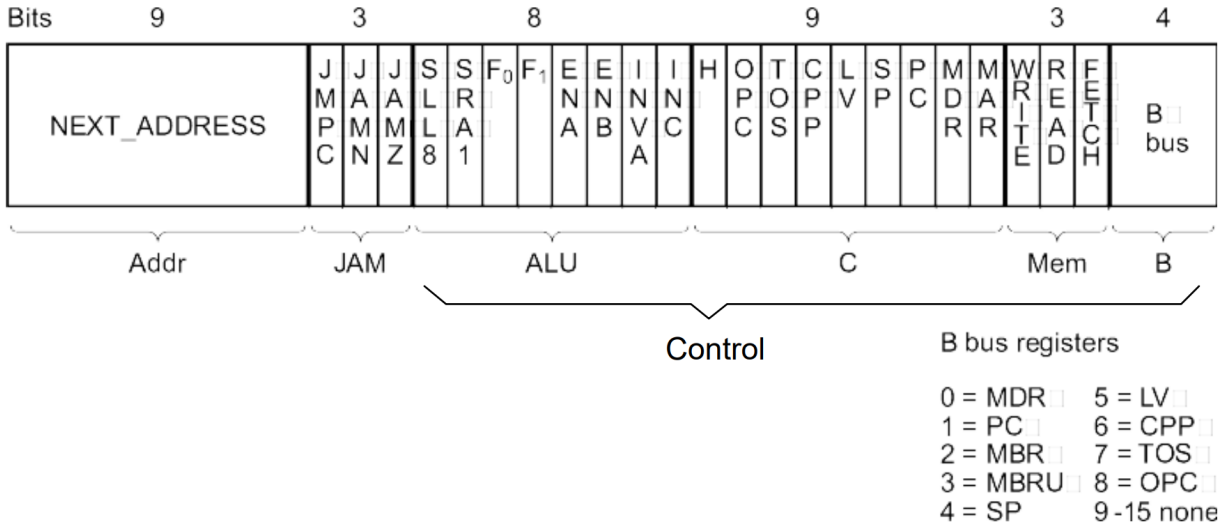
- 9 segnali di scrittura del **Bus C**.
- 9 segnali di attivazione dei registri sul **Bus B**.
- 8 segnali per controllare le funzioni di **ALU** e **shift**.
- 2 segnali per lettura/scrittura di **MAR/MDR**.
- 1 segnali di fetch per **PC/MBR.**

Questi segnali specificano il comportamento della CPU in un ciclo, che nella nostra architettura **un ciclo di data path coincide con il periodo di clock. Però un**

ciclo di data path non corrisponde a un'istruzione ISA per fare in modo che questo avvenga occorrono più cicli.

La **sequenza di cicli di data path** necessari per **un'istruzione ISA** prende il nome di **microprogramma** si divide in **microistruzioni**.

Come si compone una microistruzione? Attraverso tre parti: **Control**, **Address** e **JAM.**



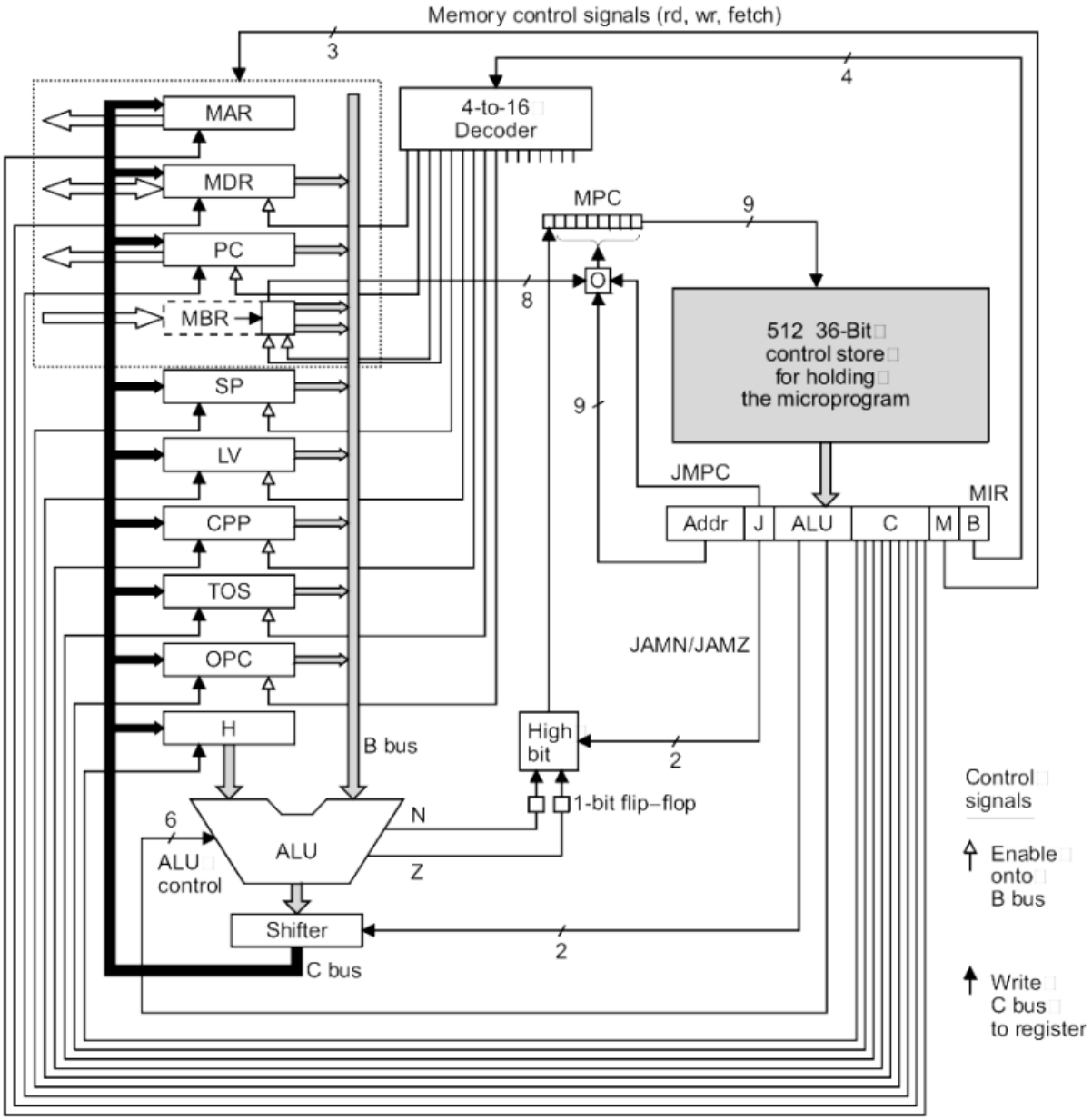
Per ridurre il numero di bit di controllo la nostra microarchitettura utilizza un **decoder** che con solo 4 bit specifica quale dei 9 registri di controllo sul Bus B attivare.

Abbiamo quindi 9 bit per **l'indirizzo**, 3 bit per il **JAM** e 24 bit per il **controllo.** Con un totale di **36 bit.** Solamente una parte delle 2^{36} possibilità di combinazione sarà usate.

Unità di controllo

Abbiamo finora analizzato la parte del data path (sinistra) mentre ora ci concentriamo su quella a destra ovvero quella di controllo. Il cuore della microarchitettura è la **control store**, ovvero una piccola ROM nella quale vengono lasciate scritte le **microistruzioni** che compongono i **microprogrammi** che "traducono" le istruzioni **ISA**. Infatti la CPU può solo leggere microistruzioni, e la

istruzioni ISA sono quindi codificate in microprogrammi composti da microistruzioni per **comunicare** tra **alto** livello e **basso** livello.



La memoria del nostro esempio sopra riportato ha 512 parole con una lunghezza di 36 bit ciascuna, la memorizzazione dei microprogrammi avviene in modo diverso rispetto alla memorizzazione dei programmi assembler.

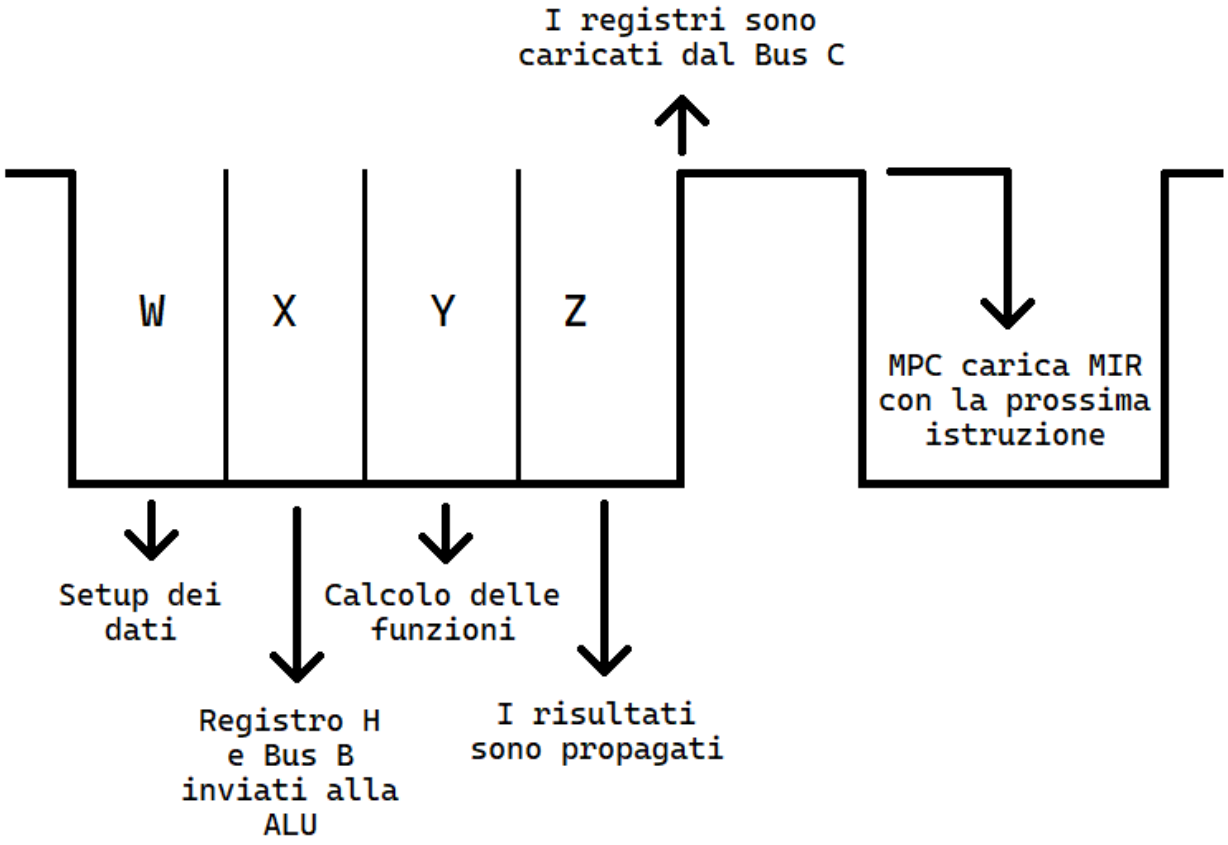
- Le **istruzioni ISA** vengono eseguite nello **stesso ordine** nel quale sono **memorizzate**. Infatti il PC (Program Counter) si incrementa. (tralasciando i salti condizionali).
- I **microprogrammi** sono più **flessibili** in quanto sequenze di microistruzioni. Infatti bisogna sempre **specificare** la **successiva microistruzione**, lasciando uno spazio apposito all'interno di esso con i 9 bit di **NEXT_ADDRESS.**

Il control store è interfacciato con la CPU tramite due registri:

- **MPC** (Micro Program Counter) che specifica l'indirizzo della prossima microistruzione da eseguire. Ha una lunghezza di 9 bit.
- **MIR** (Micro Instruction Register) che memorizza la microistruzione corrente i cui bit controllano i segnali del data path. I gruppi di segnali **ALU** e **C** sono collegati direttamente al data path, **B** è connesso al data path tramite **decoder, M** sono segnali di controllo della memoria esterna.

Funzionamento della microarchitettura

Come avviene il caricamento di una microistruzione.



- All'inizio di ogni ciclo **MIR** viene caricato con il contenuto della parola indirizzata da MCP. Il caricamento avviene in tempo *w.*
- Dopo il tempo *w* il **data path inizia** il proprio **ciclo**: **H** e uno dei **registri attraverso Bus B** vengono **inviati** alla **ALU** che calcola la funzione richiesta.
- Al termine del calcolo i risultati sono propagati attraverso il **Bus C** ai registri dopo un tempo *z.*
- Subito **dopo** il **fronte** di **salita** il ciclo è **terminato**, i **risultati** sono **memorizzati** e i risultati delle operazioni sono presenti e **MPC** viene **caricato** con un **nuovo valore** e nel mentre i **bit di stato N** e **Z** vengono memorizzati su due flip-flop per essere usati in seguito.

Come avviene il caricamento dell'istruzione successiva?

- Dopo che MIR è carico e stabile ovvero dopo il fronte di discesa, il campo di 9 bit di **NEXT_ADDRESS** viene caricato in MPC.
- Viene ora controllato il campo **JAM.**

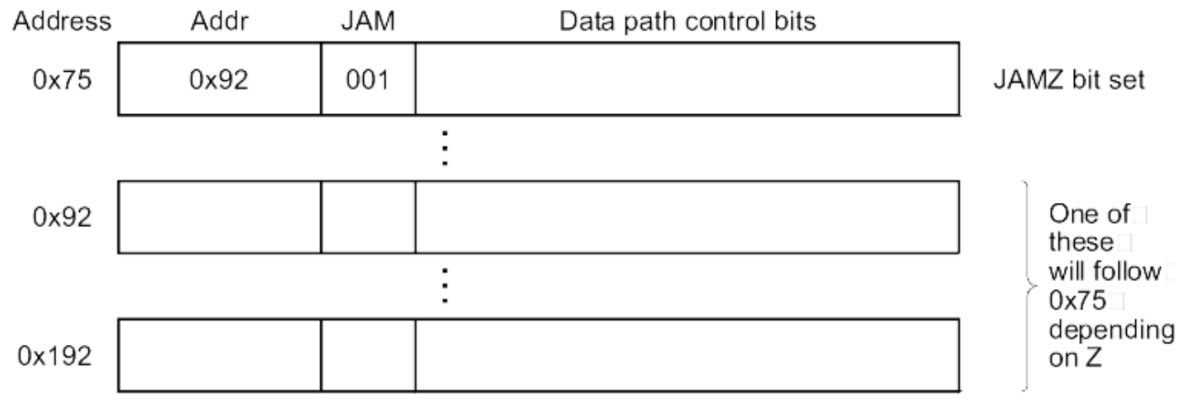
- Se tutti e **3 i bit sono a 0**, l'indirizzo è semplicemente **NEXT_ADDRESS** e viene atto altro.
- Se **JAMN è 1,** il bit più significativo di MCP viene messo in OR con **N**.
- Se **JAMZ è 1,** il bit più significativo di MCP viene messo in OR con **Z**.
- Se entrasmbi sono a 1 si fa l'OR con entrambi.

In ogni caso MPC conterrà uno dei due valori:

- **NEXT_ADDRESS**
- **NEXT_ADDRESS** con il bit più significativo in OR con 1 (ovvero sarà sempre 1).

L'importanza di JAM è per i **salti condizionali** a seconda del valore di **Z** ed **N**.

> $$F = (JAMZ \wedge Z) \vee (JAMN \wedge N) \vee (NEXT - ADDRESS[8])$$



Nell'immagine sopra riportata a seconda dello stato di Z, se è a 0 la prossima sarà 0x92 se invece è a 1 sarà 0x192.

Per finire abbiamo il bit 3 di JAM chiamato **JAMC** il quale quando è a 1 i bit di **NEXT_ADDRESS** valgono tutti 0 perchè viene eseguito l'OR bit-a-bit tra MBR e gli 8 bit meno significativi, questo segna **l'inizio di un nuovo microprogramma.**

Introduciamo ora una nuova terminologia **MAL** (Micro Assembly Language) infatti scrivere tutti e 36 i bit per ogni microistruzione risulterebbe impossibile per questo

avviene una **traduzione** da **binario** a **MAL**, nella nostra notazione tutto quello che avviene in un ciclo deve essere scritto in una riga.

Ad esempio se volessimo leggere il contenuto di SP sul Bus B, incrementarlo di 1 tramite ALU, memorizzare il risultato in SP, avviaree una lettura e mettere il prossimo indirizzo come 122 dovremmo scrivere:

```
ReadRegister = SP, ALU = INC, WSP, Read, NEXT_ADDRESS = 122
```

Ancora più semplicemente:

```
SP = SP + 1; rd; goto 122
```

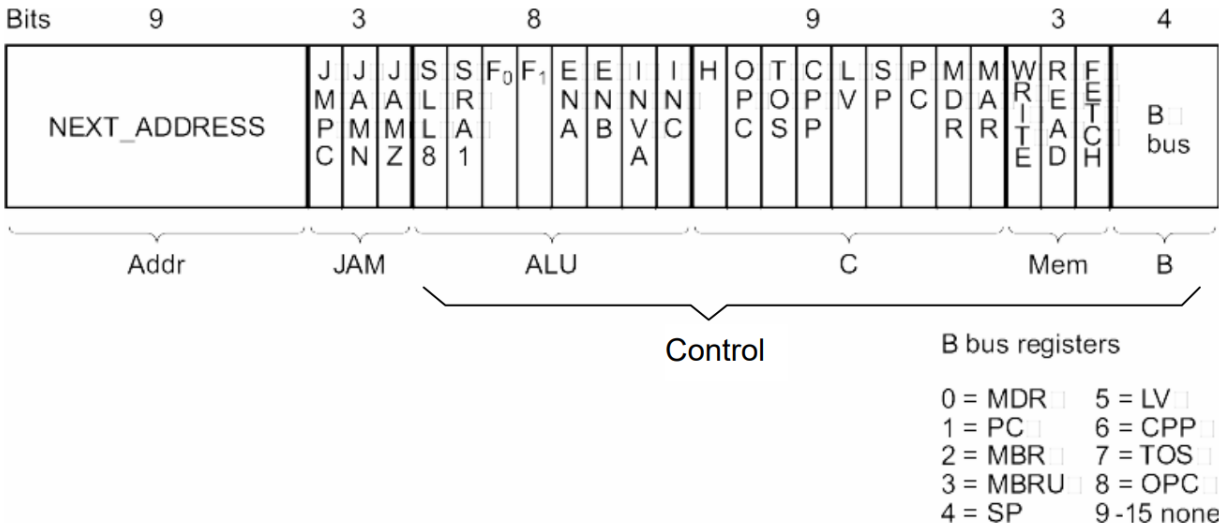
Fasi del ciclo Fetch Decode Execute

Prima di tutto abbiamo la fase di **fetch** nella quale avviene il “setup” dei registri infatti prima di tutto il **PC** passa il suo valore alla **MAR**, a seconda del segnale presente sul bus di controllo attivato dalla **CU, MAR andrà a leggere o scrivere su quel registro.** Dopo la lettura MAR deposita grazie al bus data il contenuto della locazione in memoria su **MDR** quest’ultimo passa il suo contenuto all’IR che è “compreso” dentro il **MBR** a questo punto si incrementa **PC.**

A questo punto avviene la fase di **decode** la quale associa l’attuale istruzione dentro la sua pool di istruzioni, una specie di RAM come disposizione quindi indirizzo e valore.

La fase di **execute** avviene quando il valore viene passato nuovamente alla MAR la quale grazie alla CU capisce nuovamente se deve leggere o scrivere, il valore viene passato al registro accumulatore ovvero H (holder) il quale aspetterà il ciclo successivo per entrare assieme a un altro dato dentro la ALU.

Ad ogni periodo di clock viene **svolta una fase del ciclo**, ma questo non limita la possibilità di svolgere altri cicli che non occupino le stesse celle di memoria.



Esercizi secondo esonero

```
MDR = TOS = MDR + H; wr; if(Z) goto 0x11F; else goto 0x01F
```

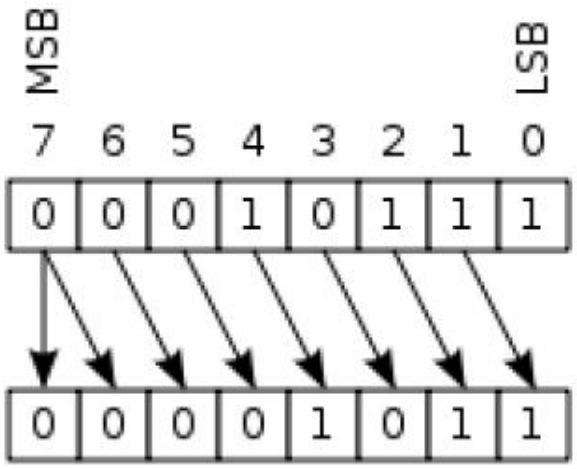
Le cifre 0x11F e 0x01F sono rispettivamente 100011111 mentre il secondo 000011111.

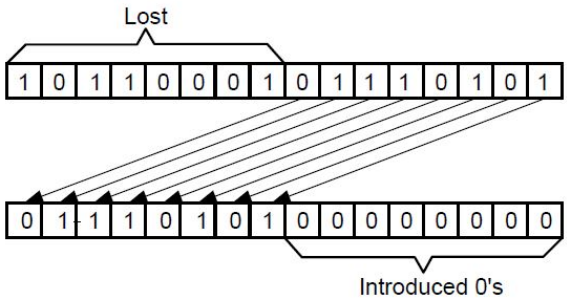
- Qual'è l'indirizzo della prossima **microistruzione** da caricare in **MIR**?
  - La prossima istruzione da caricare in MIR sarà 0x11F se **MDR + H è uguale a 0** allora Z è 1, dato che quest'ultimo tiene traccia del risultato della ALU assieme a N (che tiene traccia del risultato negativo infatti è 1 quando il risultato è negativo ed è 0 se il risultato è positivo).
- Che tipo di salto è?
  - Il salto può essere **condizionato**, **incondizionato** e **a molte vie**, in questo caso essendo presenta la if è condizionato. (Se JAMC è 1 è un salto a molte vie).
- Quale funzione è richiesta all'ALU e allo shifter?
  - ALU deve calcolare la funzione **A + B** (seguendo la tabella allegata 11111000). Lo shifter non deve far nulla (se c'è uno shift avremmo visto MDR + H **<< 8**).
- Registri dove memorizzare il valore calcolato dalla ALU?

    - In questo caso abbiamo una dopia destinazione ovvero **MDR** e **TOS**.
- Quale operazione va in memoria?
    - In memoria l'operazione è quella di **write** (scritta come wr).
- Quale registro operando?
    - Il registro è **MDR** sul Bus B.
- In **NEXT_ADDRESS** andiamo a mettere il valore contenuto nel ramo **ELSE,** sempre minore il valore di 256 quindi in questo caso 000011111 ovvero 0x01F.
- in JAM andiamo a mettere 001 dato che si dispone in questo modo C - N e Z quindi 001.
- Lo shifter ALU invece è 00.
- La ALU contiene la somma di **A + B** quindi 11111110.
- In memoria (M) 100 dato che come in JAM ci sono **tre bit di istruzioni** write, read e fetch.
- Il registro C il contenuto di TOS e MDR: 00**1** (TOS) 0000 **1**0 (MDR). Ovvero i bit sono a 1 in base ai registri a cui si è fatto "accesso".
- In B 0000 perchè dobbiamo vedere quale registro è stato usato ALU nell'espressione, inquesto caso viene "attivato" MDR + H quindi confrontandoci con la tabella 0000. Nel caso in cui non ci sia questo registro lasciamo 0000 dato che non verrà usato il Bus B verrà ignorata questa sezione.

Si specifica **rd** o **wr,** quando si comunica sulla memoria. Quando andiamo a mettere a 1 una casella in cui andremo a salvare il risultato, nel controllo.

Sommando un numero binario a se stesso, raddoppia, shiftando di 1 tutti i numeri a destra. Si imposta il **MSB** come il bit di segno.





Queste sono le due tipologie di shift presenti in questa architettura.

Funzionamento dei salti condizionali

Se abbiamo un istruzione usa un if, avrà anche un ramo else usato per specificare il **NEXT_ADDRESS,** il ramo if con un registro fa un OR con il numero così da fare il salto di registri.

Esercizio

Verificare che tutti i bit di un registro siano a 1, in codice MAL:

i00. `TOS = 0; goto i01`

i01. `Z = OPC; if(Z) goto fine else goto i02` (scrivere Z = OPC non è una assegnazione per una determinata operazione, vogliamo solo fare un salto condizionale)

i02. `N = OPC; if(N) goto incrementa else goto shift`

incrementa. `TOS = TOS + 1; goto shift`

shift. `H = OPC; goto i03`

i03. `OPC = OPC + H; goto i01`

fine. `goto fine`

Ecco un esempio di programma completo:

```
// Dato un numero binario contare i numeri di 1
.label inzio 0x00
istruzione0. TOS = 0; goto istruzione 1
istruzione1. Z = OPC; if(Z) goto fine else goto istruzione2
istruzione2. N = OPC; if(N) goto incrementa else shift
incrementa. TOS = TOS + 1; goto shift
shift. H = OPC; goto istruzione3
istruzione3. OPC = OPC + H; goto istruzione1
fine. goto fine
```

Tradotto da un compilatore MAL troviamo:

```
istruzione1. Z = OPC; if(Z) goto fine else goto istruzione2
istruzione1 in HEX: 009140008 -> 000000001 001 00010100 0000000
NEXT_ADDRESS (9bit): 0000 0000 1 // indica la prossima istruzion
JAM (3bit): 001 // ovvero è attivo su C N Z il bit di Z.
ALU (8bit): 00010100 //
C (9bit): 000000000 // Non si fanno istruzioni sul bus C
Mem (3bit): 000 //
B (4bit): 1000 // 8 in decimale ovvero OPC
```

Come si posiziona un vettore in memoria

Per posizionare un vettore, abbiamo una posizione iniziale (esempio) 0x8000 che è riempita con la dimensione del vettore. Le posizioni successive sono caricate con i valori del vettore ovvero da **0x8000+1** a **0x8000+n**.

```
1. Leggere n: MAR = 0x8000; rd
// Dobbiamo aspettare un ciclo di clock per la fine di rd e trov
2. TOS = MDR // Letto il dato lo immagaziniamo in TOS
3. H = 0
4. MDR = H = H + 1 // Incrementiamo H e salviamo sia su H che su
5. MAR = indirizzo + H; wr
6. Se H è uguale a TOS torno al 4 altrimenti fine.
```

Ecco il codice completo del programma.

```
.label inizio 0x00

inizio. OPC = 1; goto i2 // Sequenza che costruisce l'indirizzo
i2. OPC = OPC << 8; goto i3 //  OPC = 0x100
i3. OPC = OPC << 8; goto i4 // OPC = 0x10000 di 1 troppo avanti
i4. OPC = OPC >> 1; goto i5 // OPC = 0x8000 questo perchè (vedi
i5. MAR = SP = OPC; rd; goto i6 // Assegna a SP e OPC l'indiriz
i6. H = 0; goto i7 // Imposta il contatore a 0
i7. TOS = MDR; // Si salva il contenuto di MDR in TOS ovvero n
i8. MDR = H = H + 1; goto i9 // Incrementiamo il valore di H e
                                                        // perch
                                                        // sara
i9. MAR = H + SP; wr; goto i10 // In base ad H avanza di
                                                            // c
i10. Z = TOS - H; if(Z) goto i11 else goto i8 // Confrontiamo i

i11. goto i11
```

- L’indirizzo 0x10000 equivale a 0001 0000 0000 0000 0000 che shiftato di 1 bit a sinistra equivale a 0000 1000 0000 0000 quindi 0x8000.