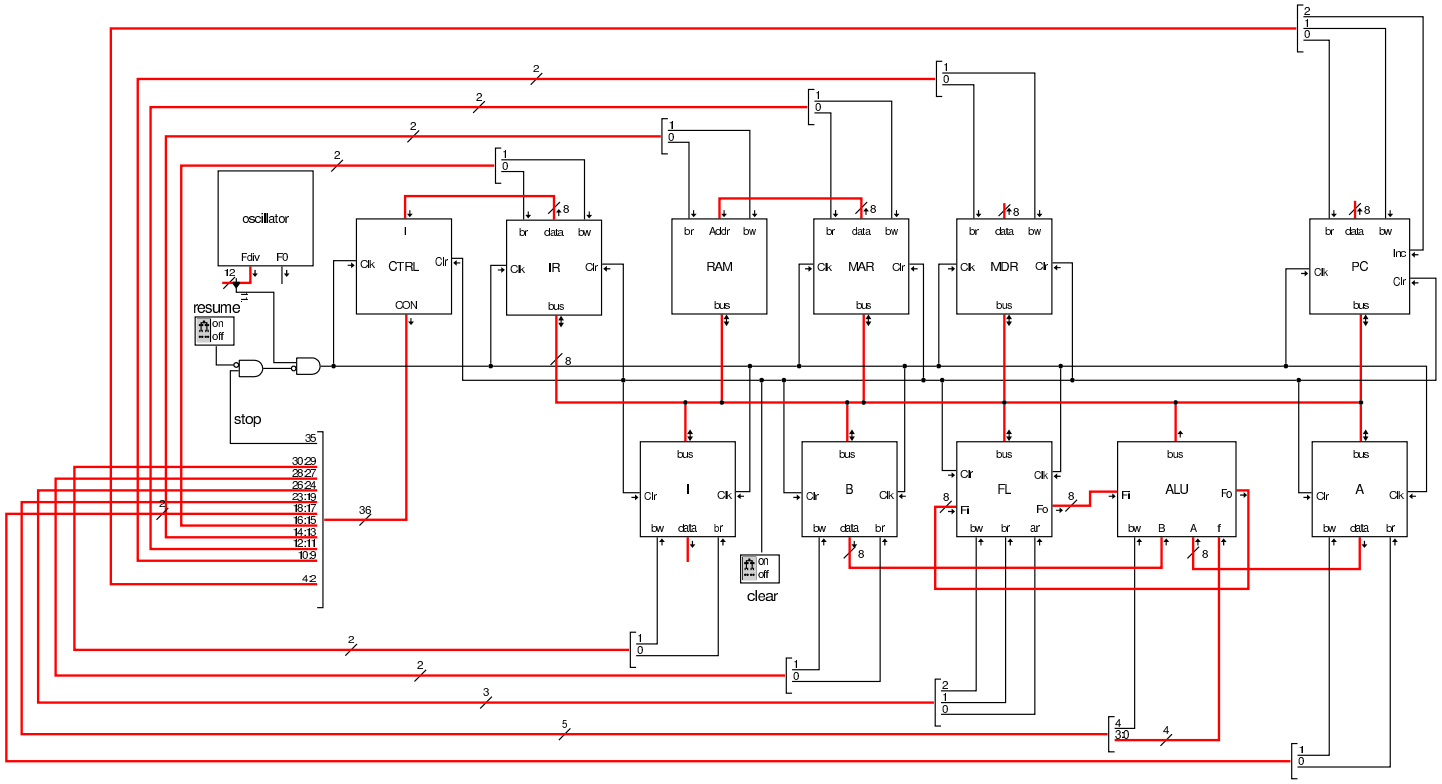
Versione E: indicatori

Nella quinta versione della CPU dimostrativa, viene aggiunto un registro per annotare lo stato degli indicatori, relativi all'esito di alcune operazioni svolte dalla ALU: riporto, segno, zero e straripamento.

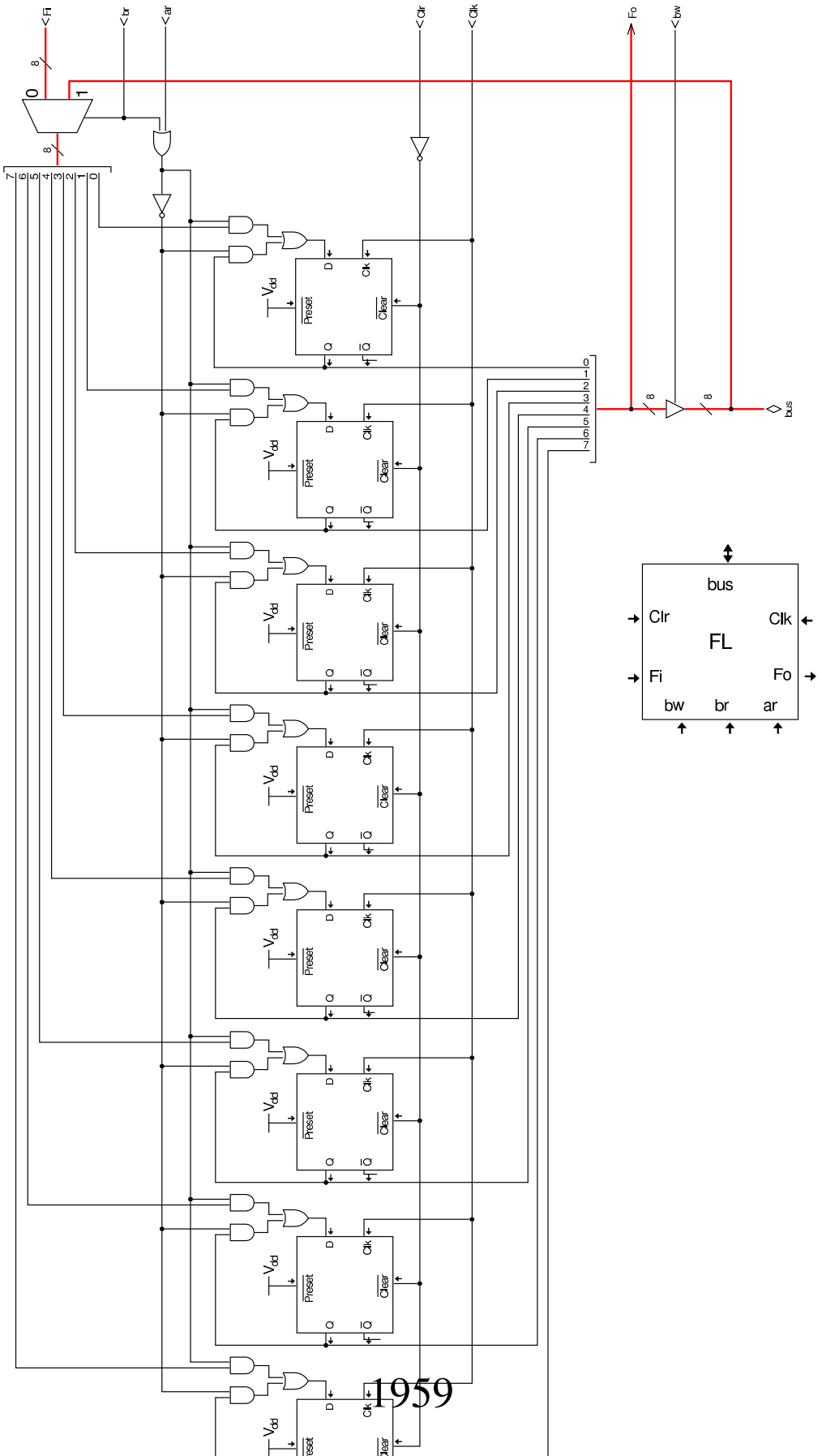
Figura u110.1. Il bus della CPU con l'aggiunta del registro ***FL*** per la gestione degli indicatori.



Come si può comprendere dagli ingressi e dalle uscite che possiede, il registro ***FL*** può immettere dati nel bus e può essere modificato leggendo dati dal bus; inoltre, può leggere direttamente dalla ALU

(ingresso ***Fi***), e per questo esiste un ingresso di abilitazione ulteriore, denominato ***ar*** (*ALU read*), mentre fornisce in ogni istante il proprio valore memorizzato alla ALU stessa (uscita ***Fo***).

Figura u110.2. La struttura interna del registro ***FL***: gli otto moduli che si vedono sono flip-flop D.



Nel codice che descrive i campi del bus di controllo, si aggiungono quelli seguenti (a parte ***fl_ar*** già apparso nella sezione precedente), i quali servono specificatamente a gestire il registro ***FL***:

```
field fl_ar[24];          // FL <-- ALU
field fl_br[25];          // FL <-- bus
field fl_bw[26];          // FL --> bus
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice:

```
op move_mdr_fl {
  map move_mdr_fl : 9;            // move MDR to FL
  +0[7:0]=9;
  operands op_0;
};
op move_fl_mdr {
  map move_fl_mdr : 10;           // move FL to MDR
  +0[7:0]=10;
  operands op_0;
};
op rotcl {
  map rotcl : 42;                 // A = A rotate carry left
  +0[7:0]=42;
  operands op_0;
};
op rotcr {
  map rotcr : 43;                 // A = A rotate carry right
  +0[7:0]=43;
  operands op_0;
};
op add_carry {
  map add_carry : 44;             // A = A + B + carry
  +0[7:0]=44;
  operands op_0;
```

```
};
op sub_borrow {
  map sub_borrow : 45;              // A = A - B - borrow
  +0[7:0]=45;
  operands op_0;
};
```

```
begin microcode @ 0
…
//
move_mdr_fl:
    fl_br mdr_bw;                   // FL <-- MDR
    ctrl_start ctrl_load;           // CNT <-- 0
//
move_fl_mdr:
    mdr_br fl_bw;                   // MDR <-- FL
    ctrl_start ctrl_load;           // CNT <-- 0
//
rotcl:
    a_br alu_f=rotate_carry_left alu_bw fl_ar; // A <-- A rot. carry l
    ctrl_start ctrl_load;                     // CNT <-- 0
//
rotcr:
    a_br alu_f=rotate_carry_right alu_bw fl_ar; // A <-- A rot. carry r
    ctrl_start ctrl_load;                      // CNT <-- 0
//
add_carry:
    a_br alu_f=a_plus_b_carry alu_bw fl_ar; // A <-- A + B + carry
    ctrl_start ctrl_load;                  // CNT <-- 0
//
sub_borrow:
    a_br alu_f=a_minus_b_borrow alu_bw fl_ar; // A <-- A - B - borrow
    ctrl_start ctrl_load;                    // CNT <-- 0
…
end
```

Figura u110.6. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

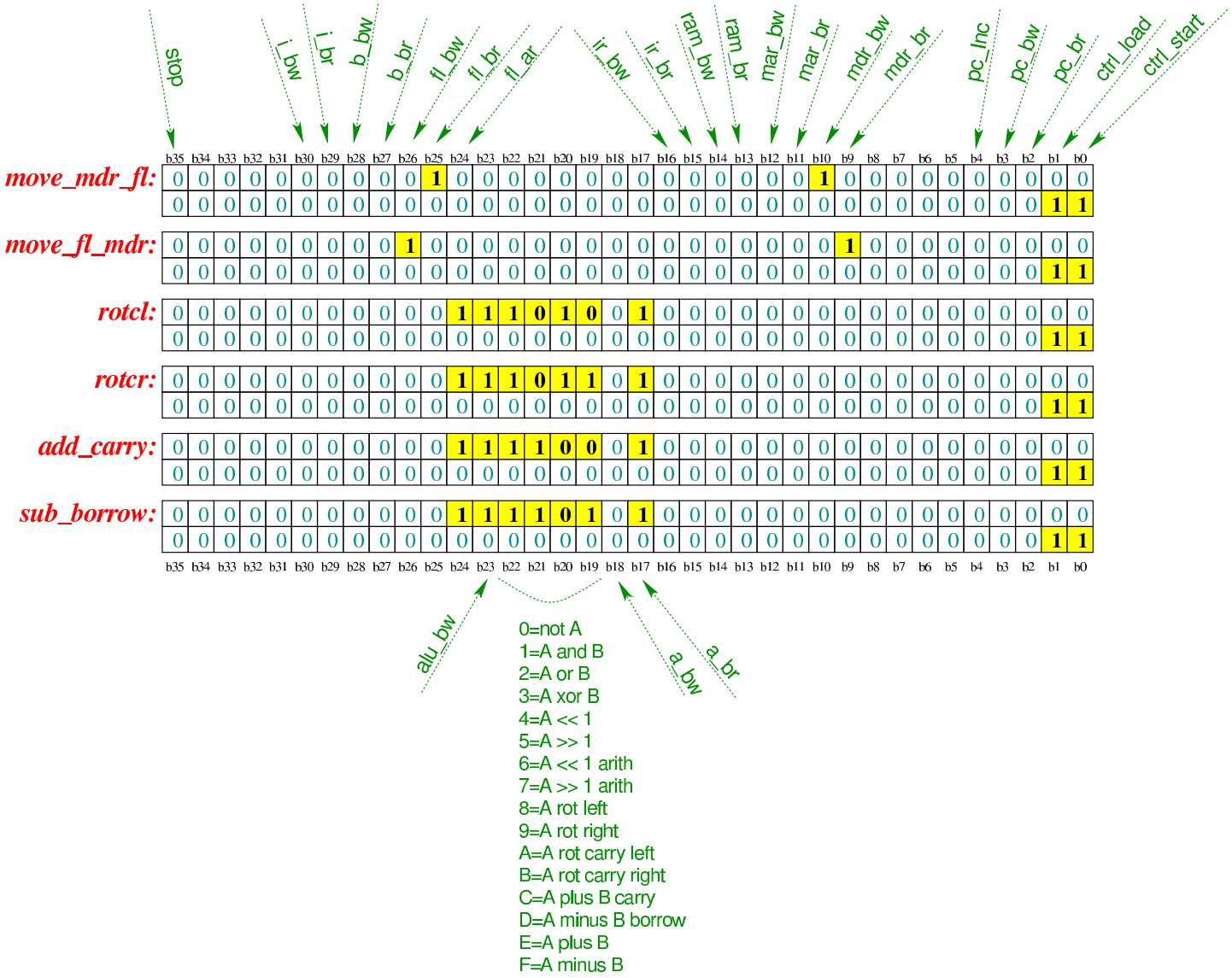


Tabella u110.7. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`move_mdr_fl`	Copia il contenuto del registro ***MDR*** nel registro ***FL***.
`move_fl_mdr`	Copia il contenuto del registro ***FL*** nel registro ***MDR***.

Sintassi	Descrizione
`rotcl`	Esegue la rotazione a sinistra del contenuto del registro ***A***, utilizzando anche l'indicatore di riporto.
`rotcr`	Esegue la rotazione a destra del contenuto del registro ***A***, utilizzando anche l'indicatore di riporto.
`add_carry`	Esegue la somma dei registri ***A*** e ***B***, tenendo conto del riporto precedente, aggiornando di conseguenza lo stesso registro ***A***.
`sub_carry`	Esegue la sottrazione ***A*−*B***, tenendo conto di un'eventuale richiesta di prestito precedente, aggiornando di conseguenza lo stesso registro ***A***.

Nelle sezioni successive, vengono proposti alcuni esempi, nei quali si sperimentano tutte le istruzioni nuove introdotte.

Istruzione «rotcl» e «rotcr»

«

Listato u110.8. Macrocodice per sperimentare le istruzioni **`rotcl`** e **`rotcr`**: si carica in memoria il valore da assegnare al registro ***A***, si eseguono cinque scorrimenti a sinistra, con l'uso del riporto e il risultato viene copiato nel registro ***B***; poi, con il valore presente in quel momento nel registro ***A***, si eseguono altri cinque rotazioni a destra, sempre con l'uso del riporto. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-rotc.gm* .

```
begin macrocode @ 0
start:
```

```
        load_imm #data_1
        move_mdr_a
        rotcl
        rotcl
        rotcl
        rotcl
        rotcl
        move_a_mdr
        move_mdr_b
        rotcr
        rotcr
        rotcr
        rotcr
        rotcr
stop:
        stop
data_1:
        .byte 160
end
```

Figura u110.9. Contenuto della memoria RAM. Le celle indicate con «xx» hanno un valore indifferente.

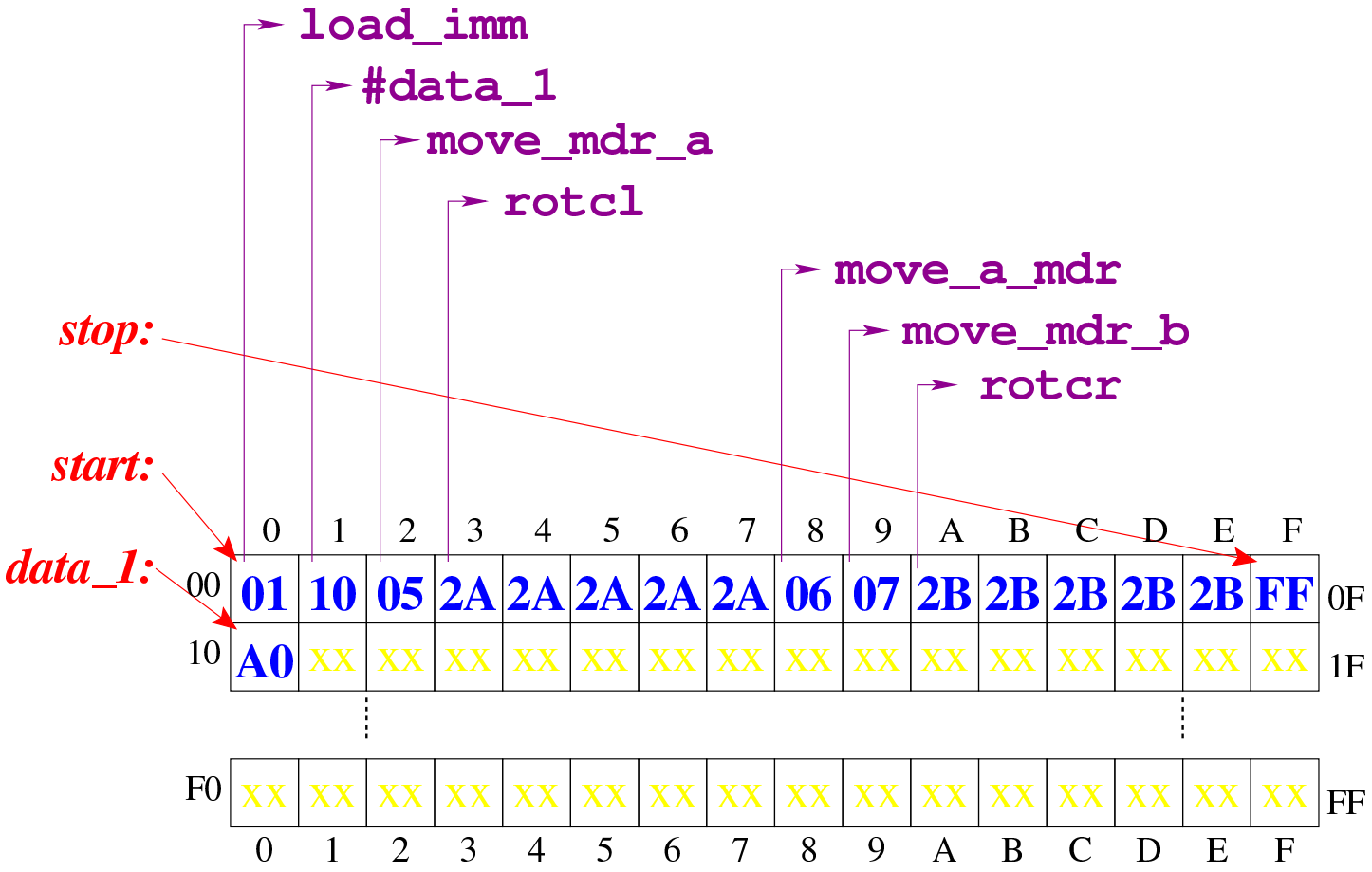
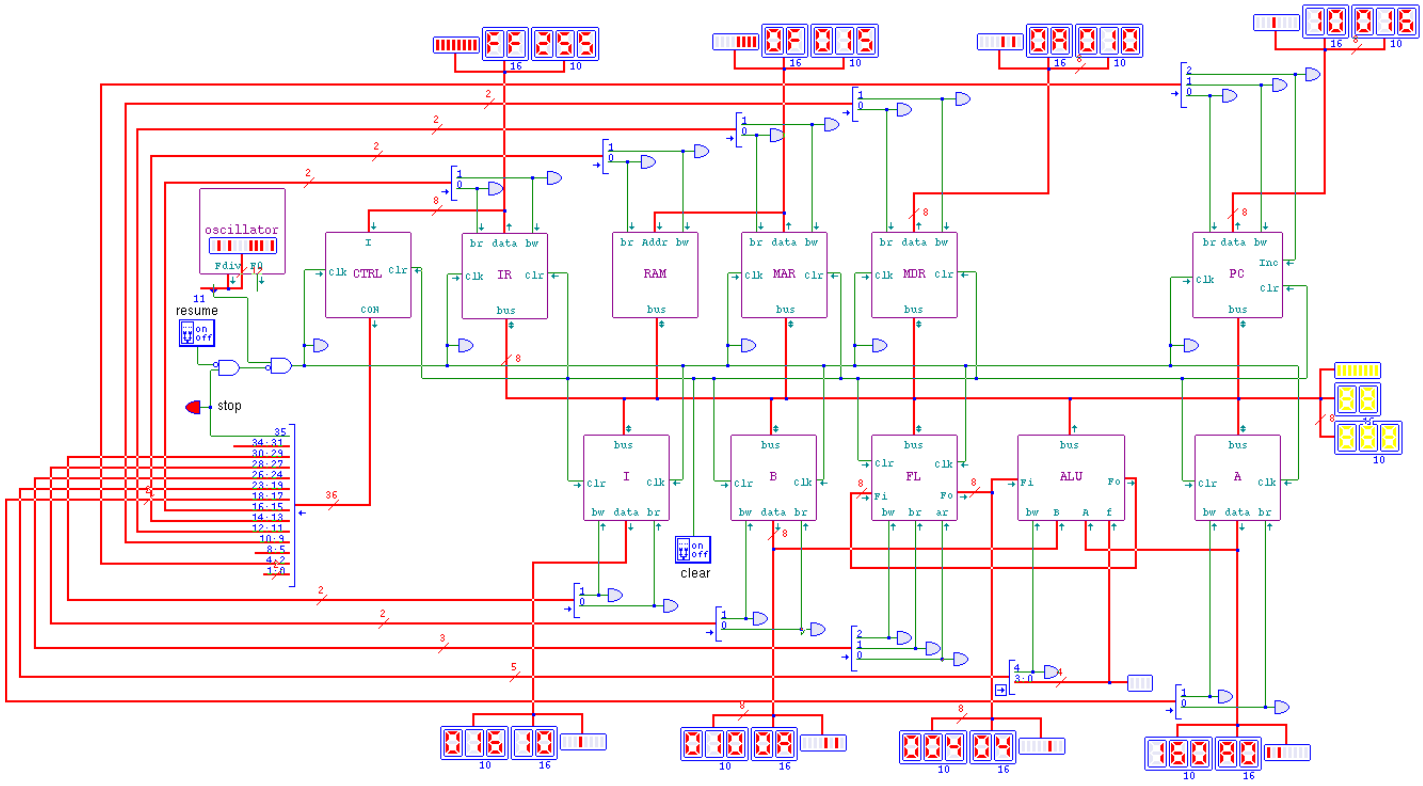


Figura u110.10. Situazione conclusiva del bus dati, dopo l'esecuzione delle istruzioni di rotazione con riporto. Video: *http://www.youtube.com/watch?v=Zl3d-Tg5C1Q*



Istruzione «add_carry»

«

Listato u110.11. Macrocodice per sperimentare l'istruzione **`add_carry`**: si vogliono sommare due numeri $12FF_{16}$ e $11EE_{16}$, necessariamente in due passaggi. Prima viene sommata la coppia FF_{16} e EE_{16}, con l'istruzione **`add`**, la quale produce il risultato ED_{16} con riporto, quindi viene sommata la coppia 12_{16} e 11_{16}, assieme al riporto precedente, ottenendo 24_{16}. In pratica, il risultato completo sarebbe $24ED_{16}$ che viene collocato in memoria dividendolo in due byte distinti. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-add_carry.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_0
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        add
        move_a_mdr
        store_imm #data_4
        load_imm #data_1
        move_mdr_a
        load_imm #data_3
        move_mdr_b
        add_carry
        move_a_mdr
        store_imm #data_5
stop:
        stop
// 0x12FF = 4863
data_0:
        .byte 0xFF
data_1:
        .byte 0x12
// 0x11EE = 4590
data_2:
        .byte 0xEE
data_3:
        .byte 0x11
data_4:
        .byte 0
data_5:
        .byte 0
end
```

Figura u110.12. Contenuto della memoria RAM prima dell'esecuzione. Le celle indicate con «xx» hanno un valore indifferente.

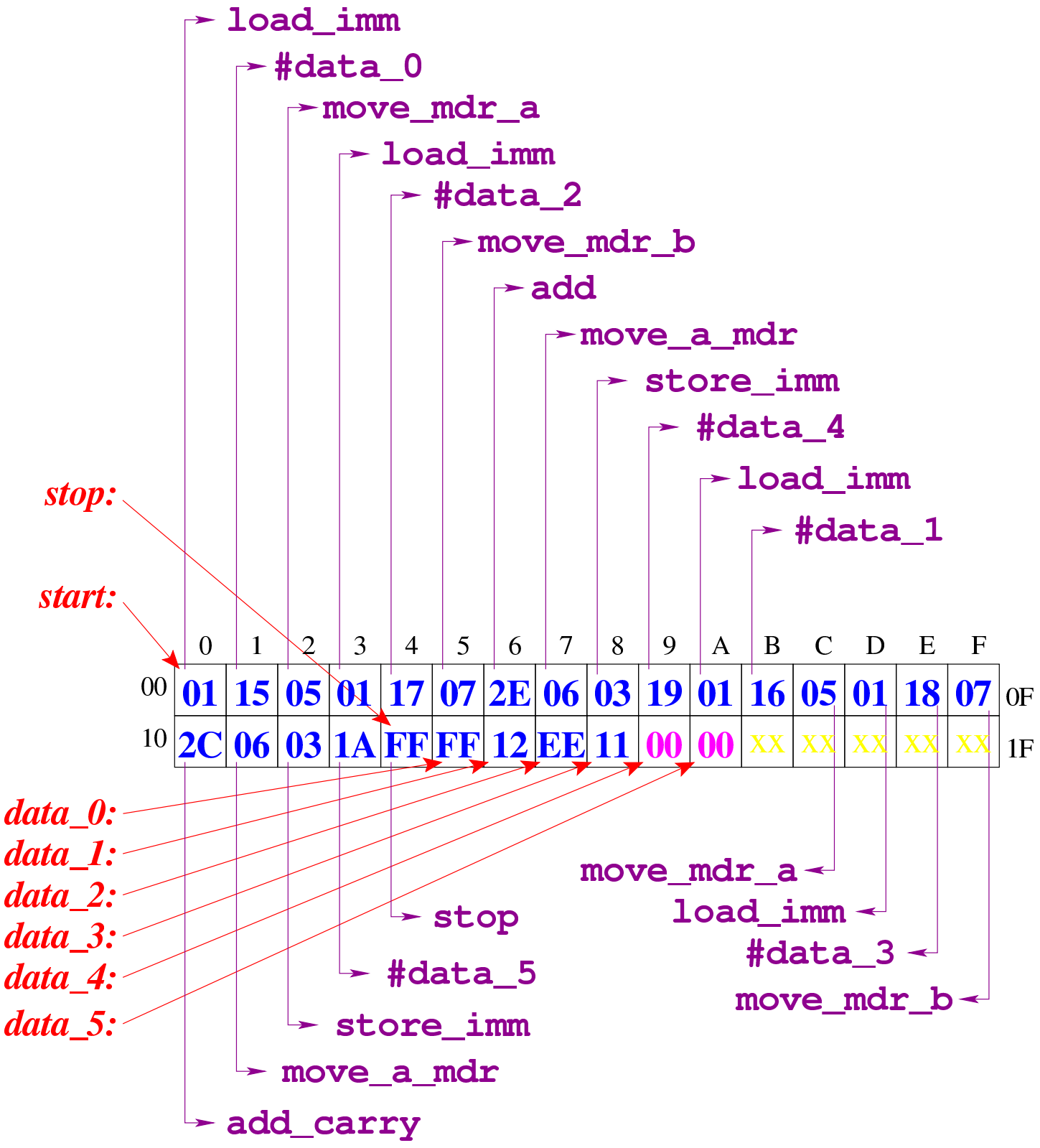


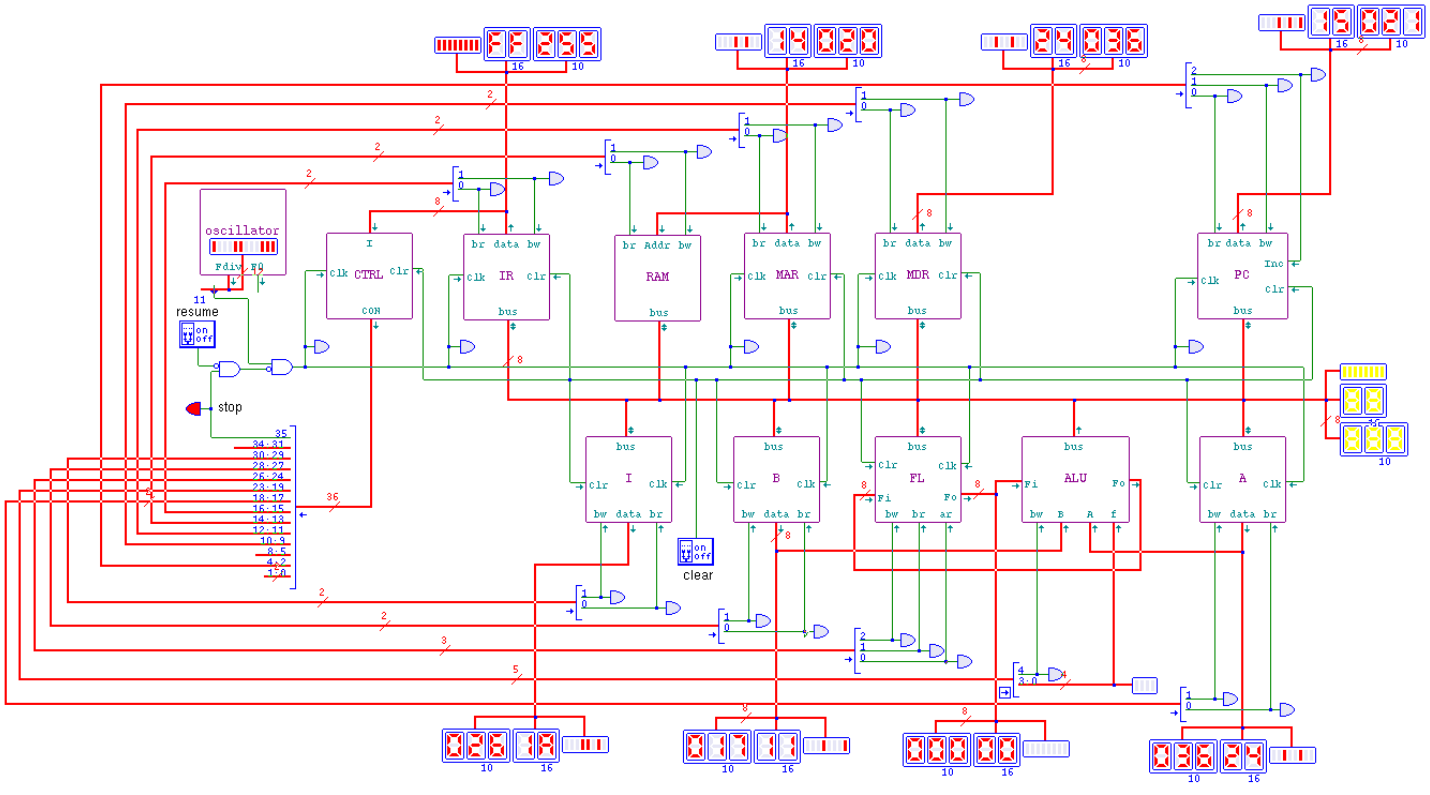
Figura u110.13. Al termine dell'esecuzione, le celle di memoria che devono contenere il risultato riportano il contenuto che si può vedere evidenziato qui.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	01	15	05	01	17	07	2E	06	03	19	01	16	05	01	18	07	0F
10	2C	06	03	1A	FF	FF	12	EE	11	ED	24	XX	XX	XX	XX	XX	1F

data_4:

data_5:

Figura u110.14. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=1Xu4MxWBwW4*



Istruzione «sub_borrow»

«

Listato u110.15. Macrocodice per sperimentare l'istruzione **sub_borrow**: si vuole eseguire la sottrazione $12EE_{16}$–$11FF_{16}$ e la si deve svolgere necessariamente in due passaggi. Prima viene sottratta la coppia EE_{16} e FF_{16}, con l'istruzione **sub**, la quale produce il risultato EF_{16} con richiesta di un prestito, quindi viene sottratta la coppia 12_{16} e 11_{16}, tenendo conto della richiesta del prestito dalle cifre precedenti, ottenendo 00_{16}. In pratica, il risultato completo sarebbe $00EF_{16}$ che viene collocato in memoria dividendolo in due byte distinti. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-d-sub_borrow.gm* .

```
begin macrocode @ 0
start:
        load_imm #data_0
        move_mdr_a
        load_imm #data_2
        move_mdr_b
        sub
        move_a_mdr
        store_imm #data_4
        load_imm #data_1
        move_mdr_a
        load_imm #data_3
        move_mdr_b
        sub_borrow
        move_a_mdr
        store_imm #data_5
stop:
        stop
// 0x12EE = 4846
```

```
data_0:
        .byte 0xEE
data_1:
        .byte 0x12
// 0x11FF = 4607
data_2:
        .byte 0xFF
data_3:
        .byte 0x11
data_4:
        .byte 0
data_5:
        .byte 0
end
```

Figura u110.16. Contenuto della memoria RAM prima dell'esecuzione. Le celle indicate con «xx» hanno un valore indifferente.

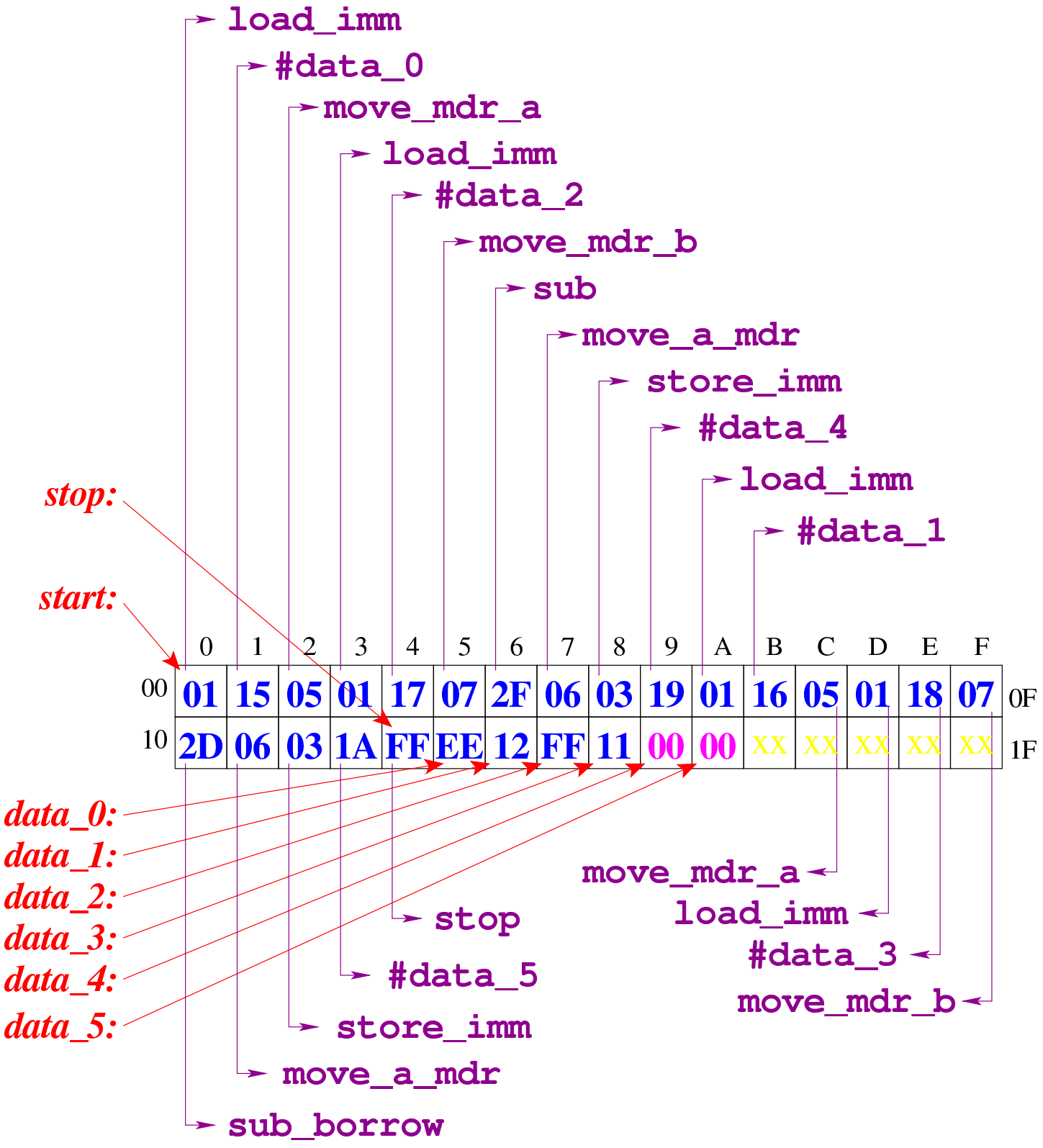


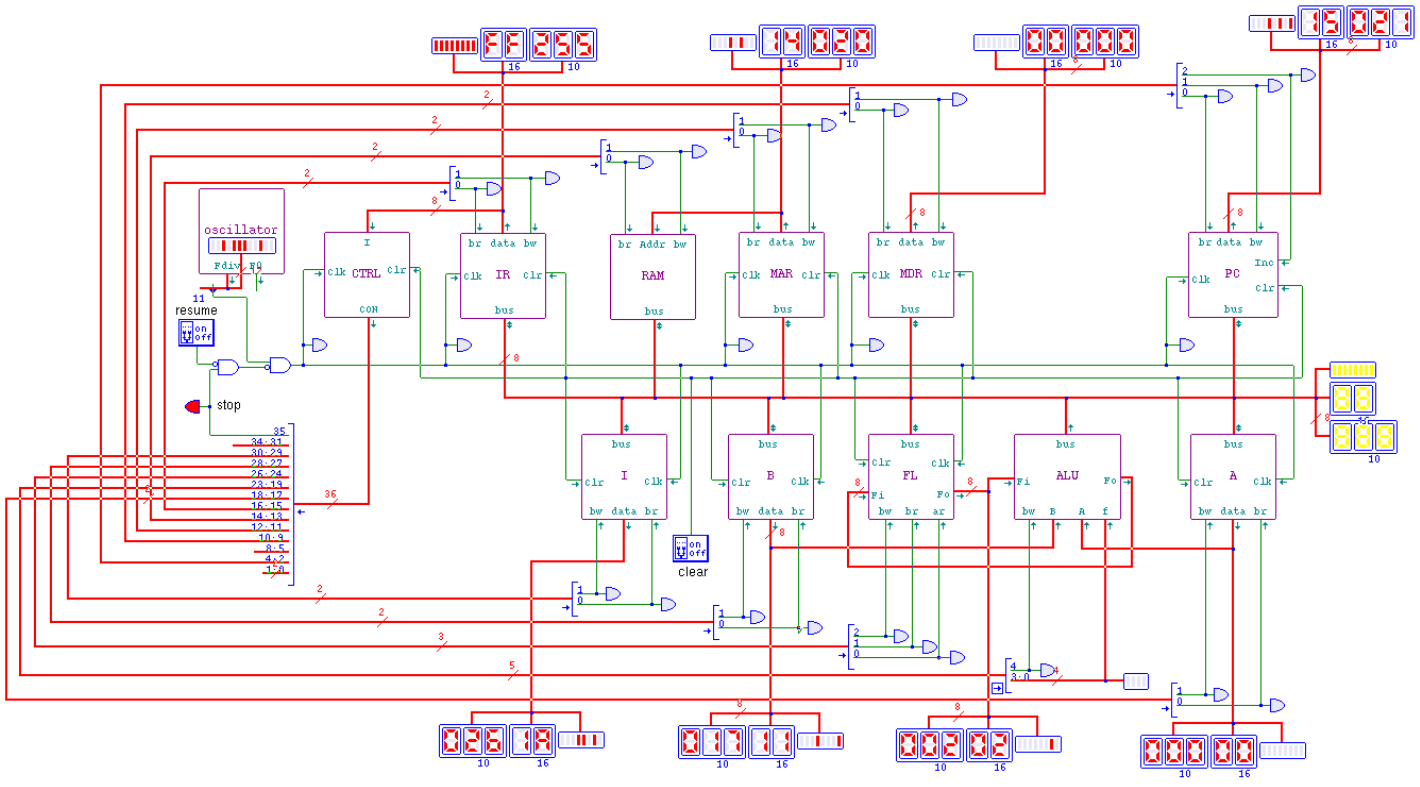
Figura u110.17. Al termine dell'esecuzione, le celle di memoria che devono contenere il risultato riportano il contenuto che si può vedere evidenziato qui.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	01	15	05	01	17	07	2F	06	03	19	01	16	05	01	18	07	0F
10	2D	06	03	1A	FF	FF	12	EE	11	EF	00	XX	XX	XX	XX	XX	1F

data_4:

data_5:

Figura u110.18. Situazione conclusiva del bus dati. Video: *http://www.youtube.com/watch?v=ofPUzdIids8*



1974