Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE-503 Estructuras de Computadores Digitales I

Tarea de Verilog: Simulación de la CPUCR

Profesor:

Lochi Yu

Estudiantes:

Jose Carlos Campos Valerio B11415

Rosario Ruiz Hernández B15867

Diciembre 2013

Introducción

En el presente trabajo se realizó una simulación en Verilog para cada instrucción de la CPUCR (con excepción de las que tienen relación con las interrupciones) con el fin de encontrar errores de su comportamiento y arreglarlos. Lo que se leerá a continuación es una bitácora que muestra el programa en ensamblador que se utilizó para probar la instrucción y el diagrama de temporización obtenido, en caso de haber encontrado errores también se muestra el código de Verilog incorrecto y el código corregido, además del diagrama de temporización luego de corregirlo.

Con el fin de tener un mayor orden en la presentación de la tarea, se mostrarán primero las instrucciones que presentaron error a la hora de la simulación, y en la segunda parte las simulaciones de las que estuvieron correctas desde el inicio.

I PARTE:

INSTRUCCIONES INCORRECTAS

CLA (CLear Accumulator)

Programa en ensamblador:

```
$0000 CLA
$0001 HLT
```

Diagrama de temporización:

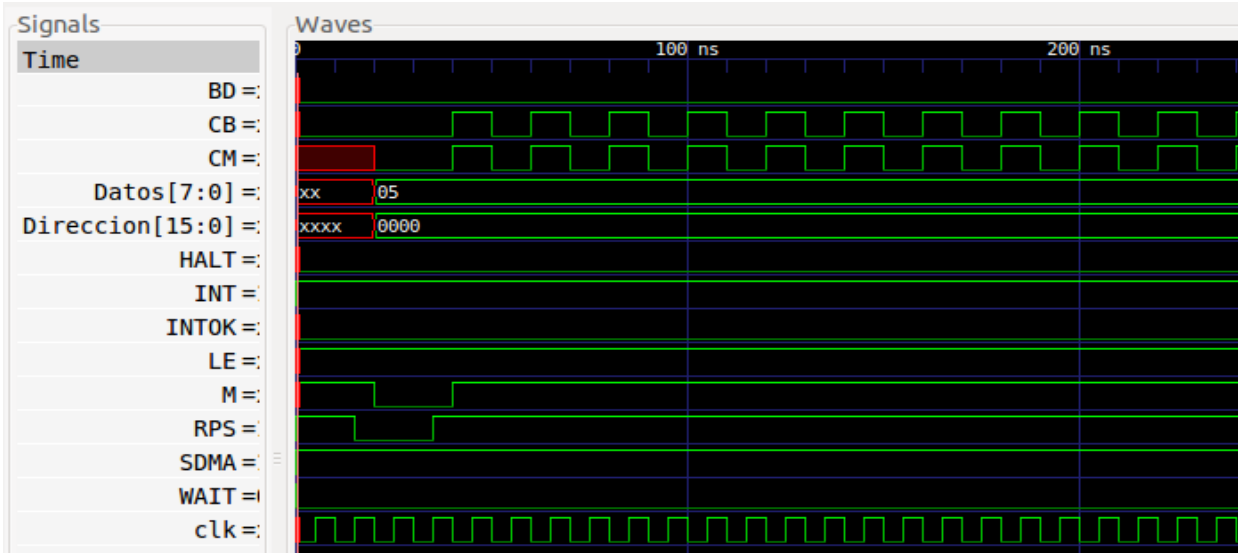

Figura 1. CLA incorrecto

Código incorrecto en Verilog: (en el módulo reg_PC)

```
CPA,LDAinm,ADDinm,TAP,TPA,CLA,INA,DCA,SUBinm,ANDinm,ORAinm,INP,OUT:
        begin
                PC<=PC;
                RDR<=RDR;
        end
```

Código correcto en Verilog:

```
CPA,LDAinm,ADDinm,TAP,TPA,CLA,INA,DCA,SUBinm,ANDinm,ORAinm,INP,OUT:
        begin
                PC<=PC+1; //le agregamos +1
                RDR<=RDR+1; //le agregamos +1
        end
```

Diagrama de temporización corregido:

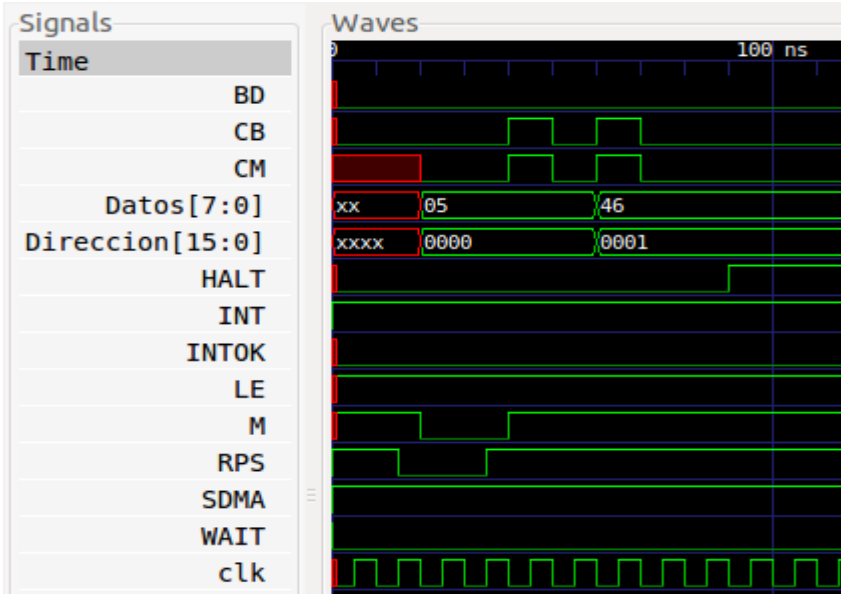

Figura 2. CLA correcto

*Como una nota, los errores que pudiesen haberse presentado en CPA, LDAinm, ADDinm, TAP, TPA, INA, DCA, SUBinm, ANDinm y ORAinm debido a este error no se presentaron porque una vez corregido esta parte del código, seguimos trabajando las instrucciones siguientes con esta corrección.

SUBinm (SUBtract inmediato)

Programa en ensamblador:

```
$0000 LDA #$0A
$0002 SUB #$06
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

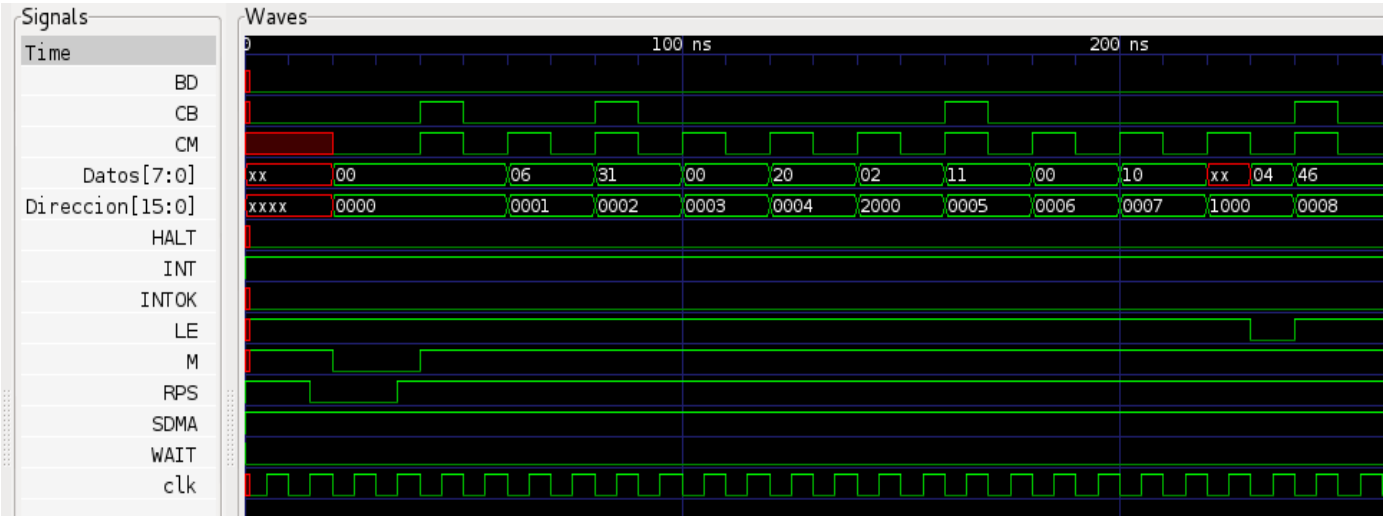

Figura 3. SUBinm incorrecto

Código incorrecto en Verilog: (en el módulo acumulador)

```
SUBinm: {BC,A}<=A+BUSDAT;
```

Código correcto en Verilog:

```
SUBinm: {BC,A}<=A-BUSDAT; // se cambio + por -
```

Diagrama de temporización corregido:

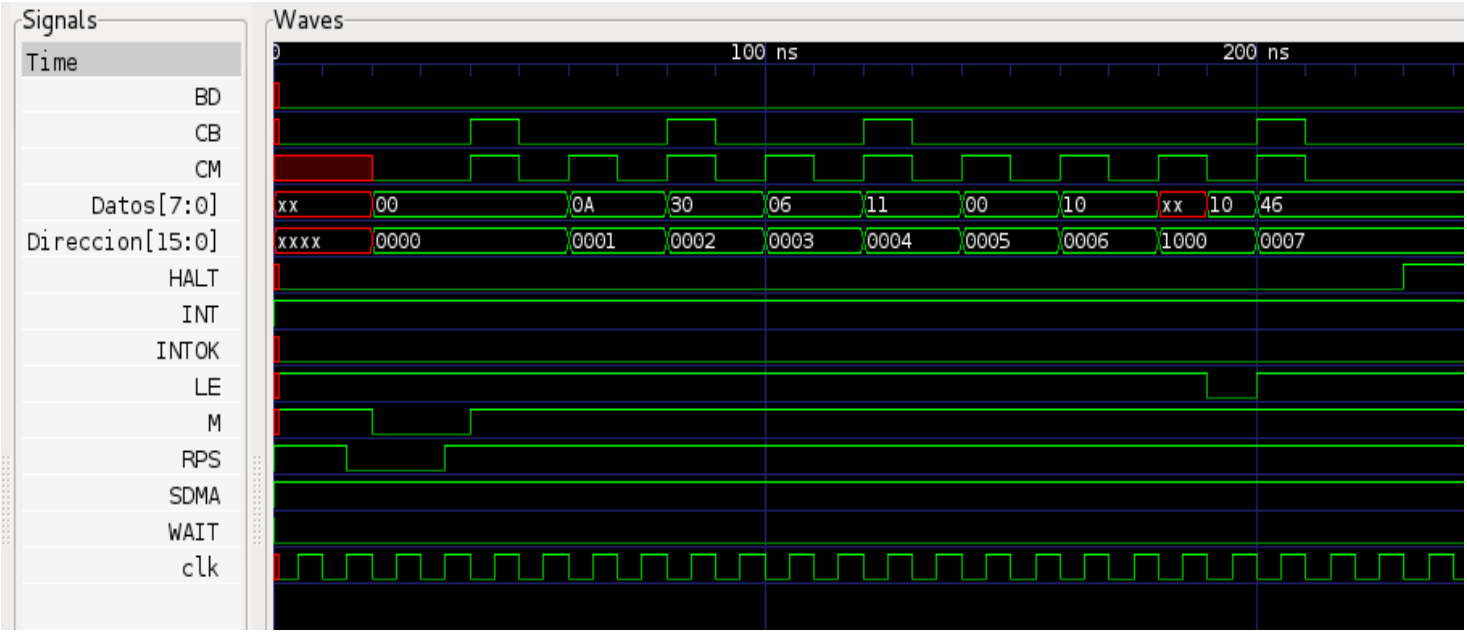

Figura 4. SUBinm corregido

STA (STore Accumulator)

Programa en ensamblador:

```
$0000 CLA
$0001 STA $1000
$0004 HLT
```

Diagrama de temporización:

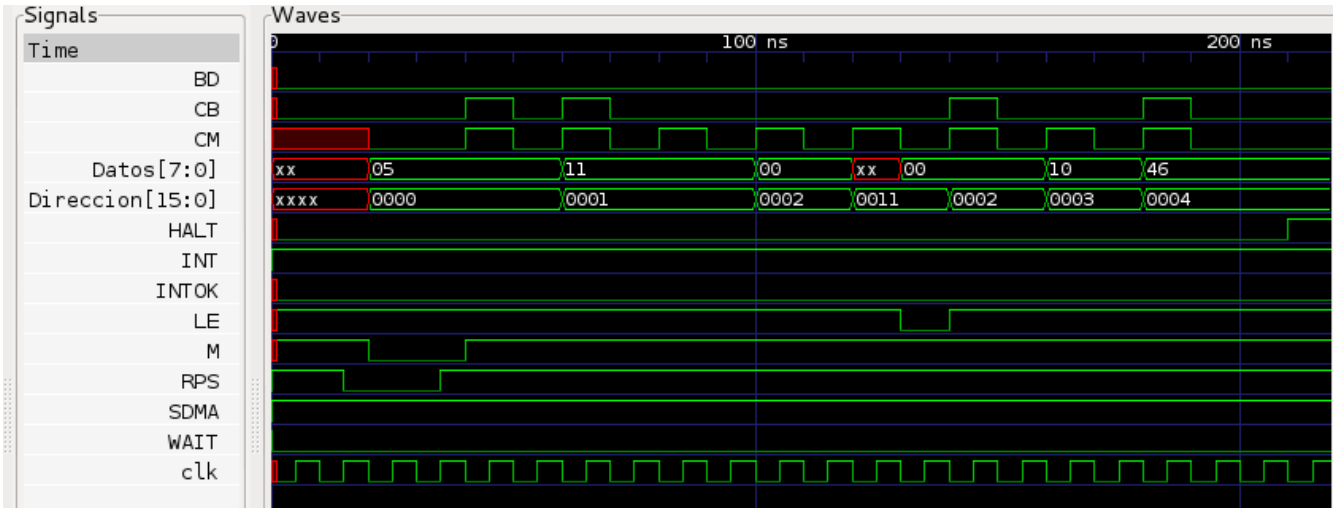

Figura 5. STA incorrecto

Código incorrecto en Verilog: (en el módulo reg_PC)

```
ROL,ROR,BCC,BCS,CLC,SEC,SEI,CLI,NOP,BNE,BEQ,BMI,BPL,STA,LDA,AND,SUB,ORA,ADD,JM
P,BVS,BVC: begin
                    PC<=PC;
                    RDR<=RDR;
                end
```

Código correcto en Verilog:

```
ROL,ROR,BCC,BCS,CLC,SEC,SEI,CLI,NOP,BNE,BEQ,BMI,BPL,STA,LDA,AND,SUB,ORA,ADD,JM
P,BVS,BVC: begin
                    PC<=PC+1; //le agregamos +1
                    RDR<=RDR+1; //le agregamos +1
                end
```

Diagrama de temporización corregido:

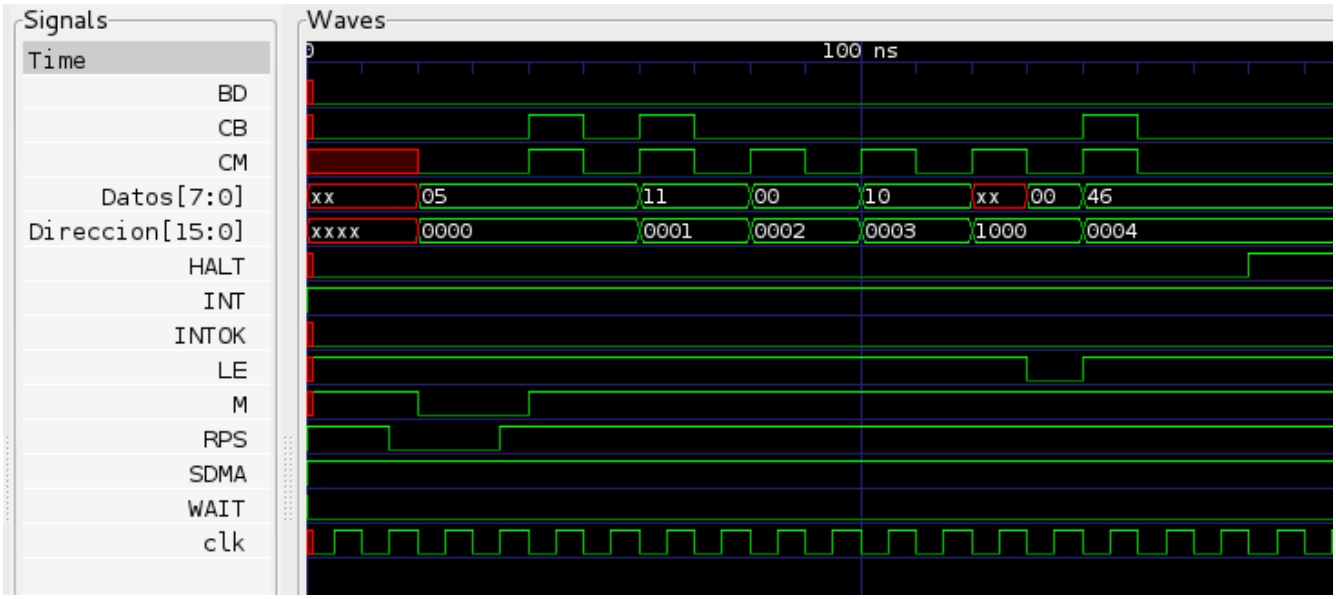

Figura 6. STA correcto

*Como una nota, los errores que pudiesen haberse presentado en ROL, ROR, BCC, BCS, CLC, SEC, SEI, CLI, NOP, BNE, BEQ, BMI, BPL, LDA, AND, SUB, ORA, ADD, JMP, BVS, BVC debido a este error no se presentaron porque una vez corregido esta parte del código, seguimos trabajando las instrucciones siguientes con esta corrección.

BCC (Branch if Carry Clear)

Programa en ensamblador:

Con el fin de saltar:

```
$0000 CLC
$0001 LDA #$FE
$0003 ADD #$01
$0005 BCC $0005
$0007 STA $1000
$0010 HLT
```

Con el fin de no saltar:

```
$0000 CLC
$0001 LDA #$FF
$0003 ADD #$01
$0005 BCC $0005
$0007 STA $1000
$0010 HLT
```

Diagrama de temporización:

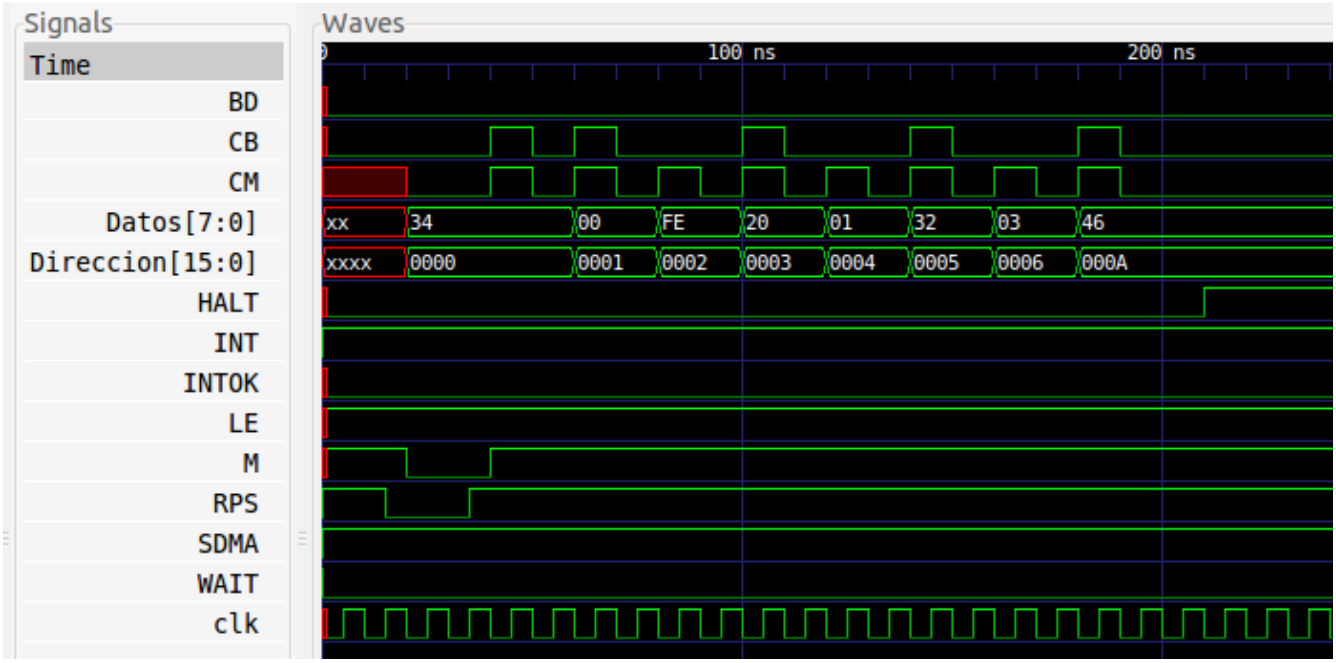

Figura 7. BCC cuando salta posiciones.

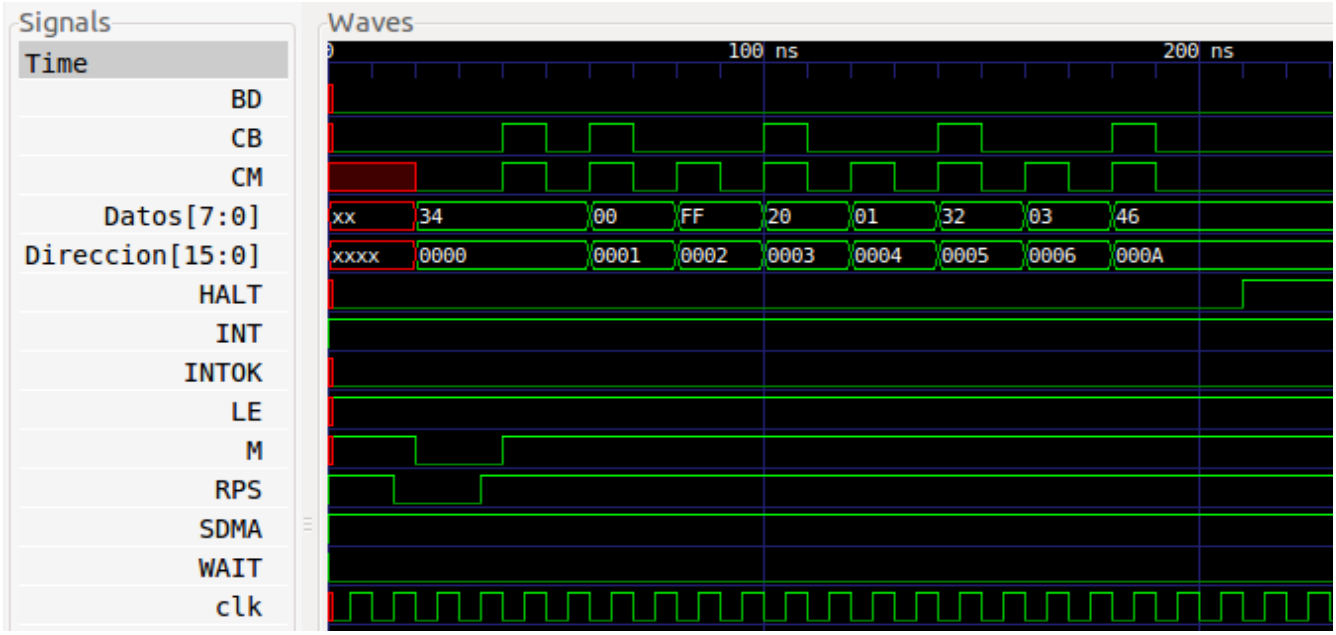

Figura 8. BCC cuando no salta posiciones.

Código incorrecto en Verilog: (en el módulo reg_PC)

```
BCC: begin

        PC<=BUSDAT[7] ? PC + {8'hFF,BUSDAT} : PC + {8'b0,BUSDAT};
        RDR<=BUSDAT[7] ? PC + {8'hFF,BUSDAT} : PC + {8'b0,BUSDAT};

         end
```

Código correcto en Verilog:

```
BCC: begin
        case(S[0])
                1'b1: begin //Lo diferenciamos para que si recibe un 1 (Carry no clear) siga.
                        PC<=PC;
                        RDR<=RDR+16'b1;
                end
                1'b0: begin
                        PC<=BUSDAT[7] ? PC + {8'hFF,BUSDAT} : PC + {8'b0,BUSDAT};
                        RDR<=BUSDAT[7] ? PC + {8'hFF,BUSDAT} : PC + {8'b0,BUSDAT};
                end
        endcase
end
```

Diagrama de temporización corregido:

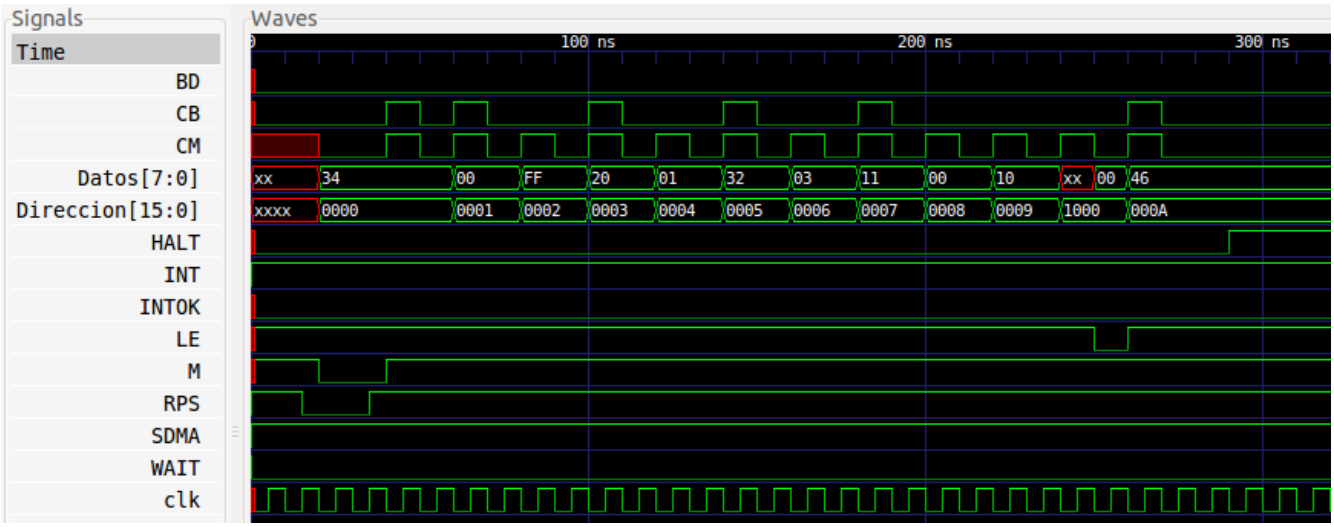

Figura 9. BCC cuando no salta posiciones.

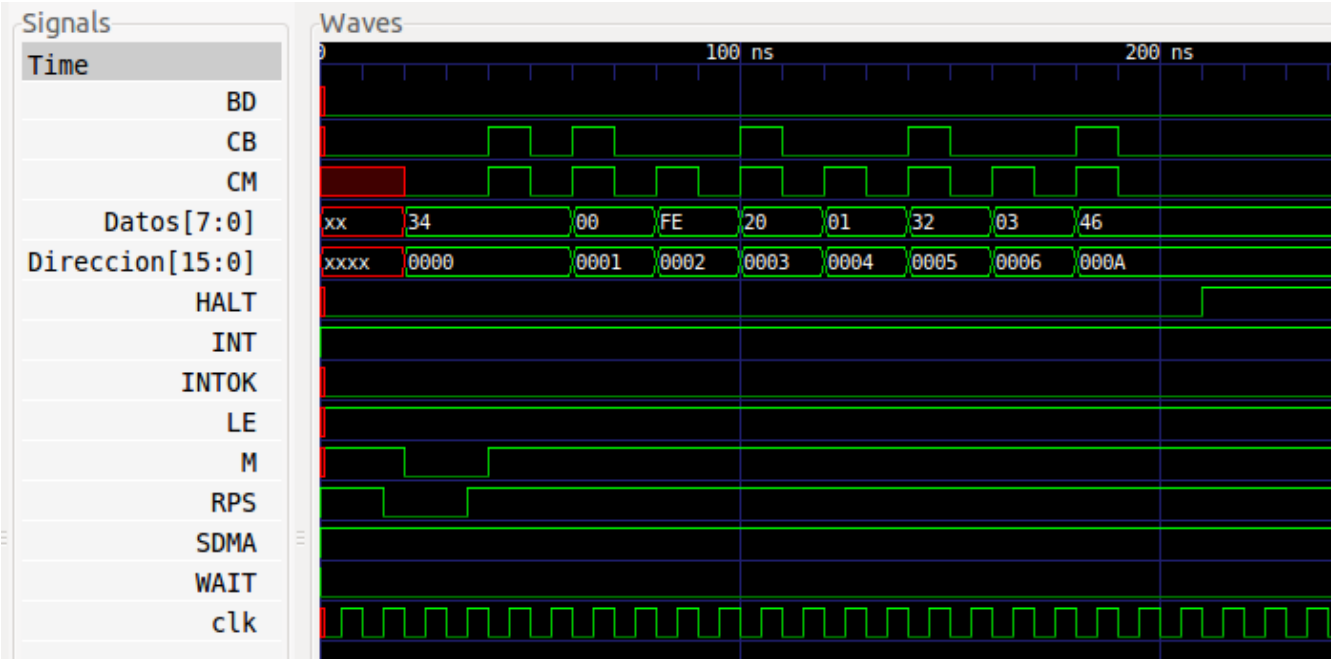

Figura 10. BCC cuando salta posiciones.

INA (INcrement Accumulator)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 INA
$0003 STA $0110
$0006 HLT
```

Diagrama de temporización:

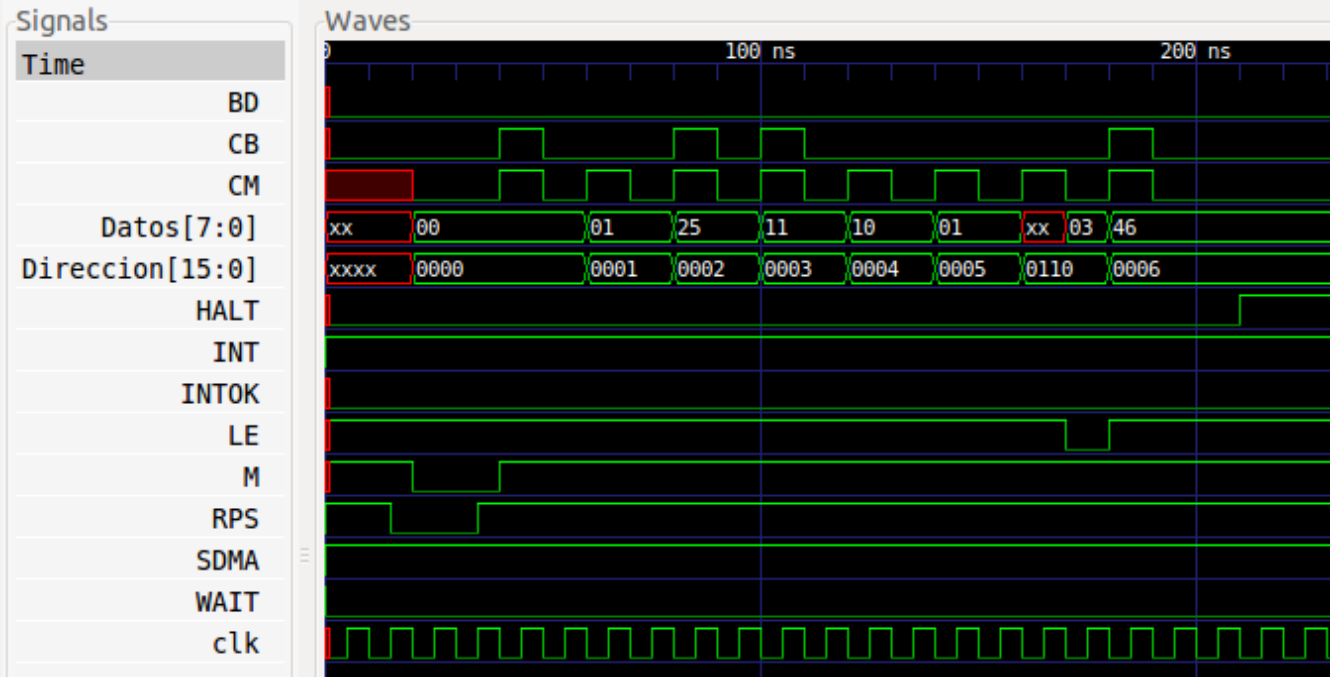

Figura 11. INA incorrecto

Código incorrecto en Verilog: (en el módulo acumulador)

INA: {BC,A}<=A+2;

Código correcto en Verilog:

INA: {BC,A}<=A+1; //se suma 1 y no 2 al acumulador

Diagrama de temporización corregido:

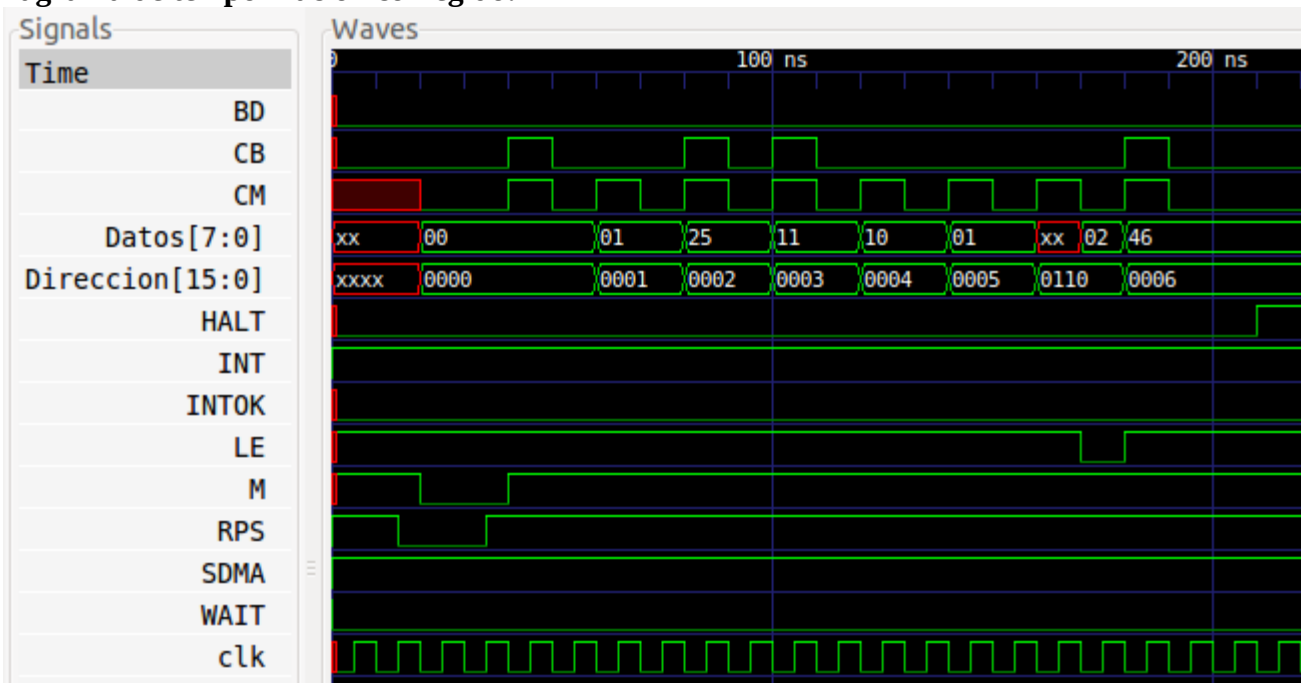

Figura 12. INA correcto

HLT (HaLT)

Programa en ensamblador:

```
$0000 HLT
```

Diagrama de temporización:

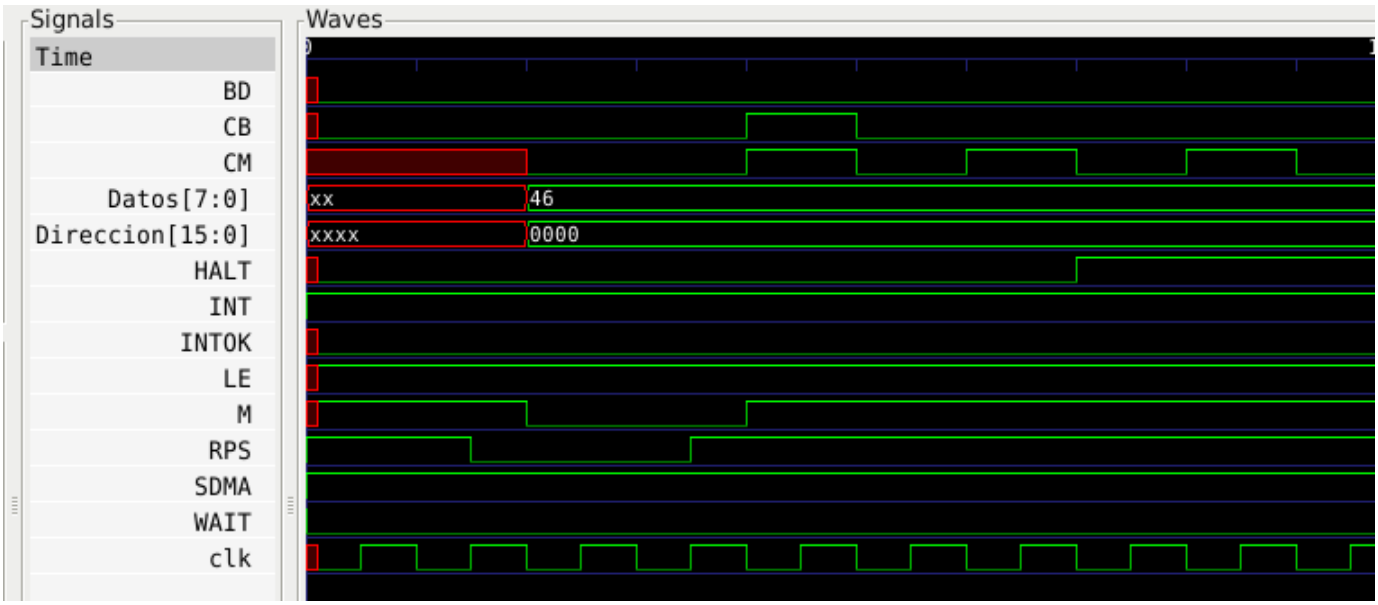

Figura 13. HLT incorrecto

Código incorrecto en Verilog: (en el módulo reg_CB)

```
always @(negedge CLK) begin
                    if(!SDMA&&((ProxEst==Estado_15)||(ProxEst==Estado_16)))
CM<=1'bz;
                    else if (!RPS) CM<=1'b0;
                    else if ((ProxEst==Estado_1)&&(EstPresente==Estado_16))
CM<=1'b1;
                    else if (ProxEst==Estado_30) CM<=CM;
                    else CM<=~(CM);
```

Código correcto en Verilog:

```
always @(negedge CLK) begin
                        if(!SDMA&&((ProxEst==Estado_15)||(ProxEst==Estado_16)))
CM<=1'bz;
                        else if (!RPS) CM<=1'b0;
                        else if ((ProxEst==Estado_1)&&(EstPresente==Estado_16))
CM<=1'b1;
                        else if (ProxEst==Estado_30) CM<=CM; //Se agrego esta linea
                        else CM<=~(CM);
```

Diagrama de temporización corregido:

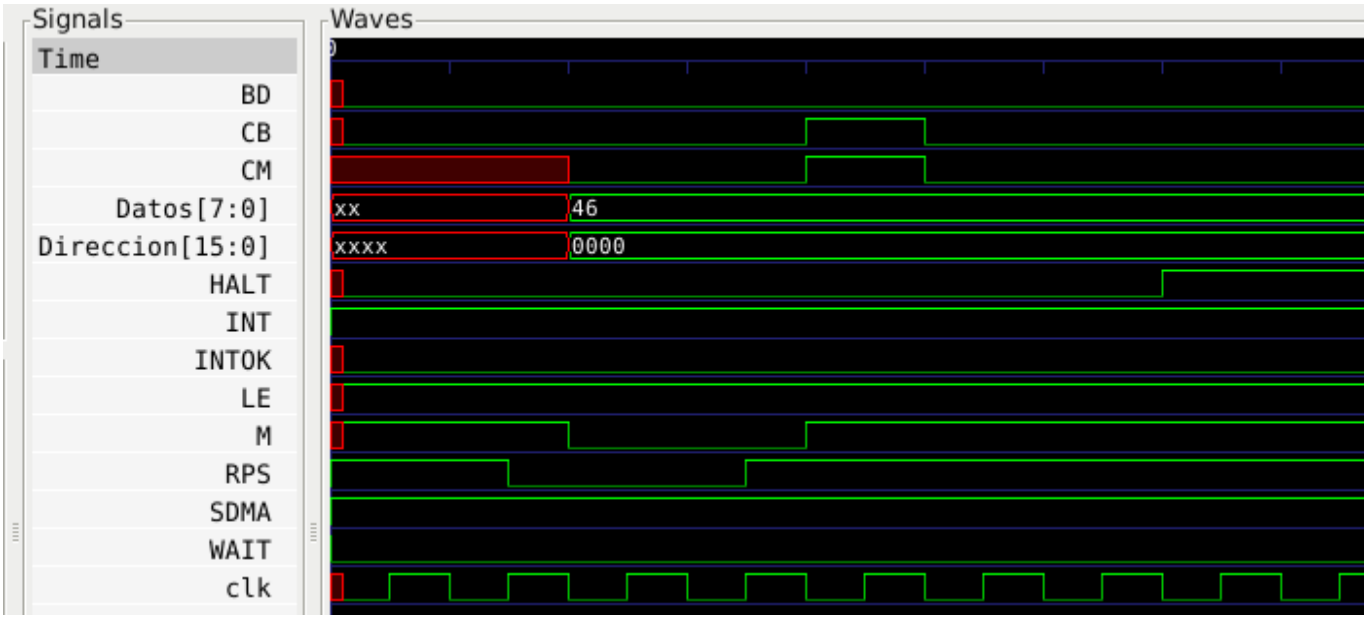

Figura 14. HLT correcto

ANDind (AND indirecto)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 AND ($1000)
$0005 STA $3000
$0008 HLT

$1000 $00
$1001 $20

$2000 $09
```

Diagrama de temporización:

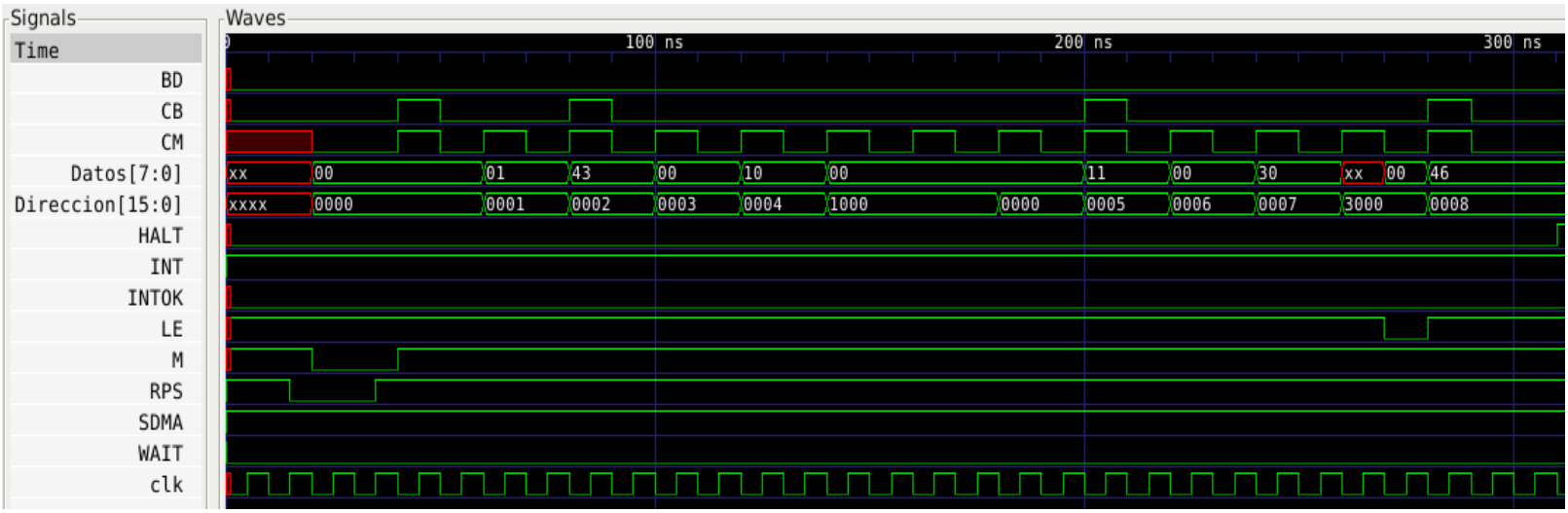

Figura 15. ANDind incorrecto

Código incorrecto en Verilog: (en el módulo reg_PC)

```
Estado_8: begin
                case(RI)
                    STA,LDA,AND,SUB,ORA,ADD: begin
                            PC<=PC;
                            RDR<=PC;
                        end
                    LDAind,SUBind,ADDind,ORAind,STAind: begin
                             PC<=PC;
                             RDR<=RDR+16'b1;
                             PB<=BUSDAT;
                            end
```

Código correcto en Verilog:

```
Estado_8: begin
                        case(RI)
                            STA,LDA,AND,SUB,ORA,ADD: begin
                                PC<=PC;
                                RDR<=PC;
                            end

                            //se agrego la instruccion ANDind a este caso

                            LDAind,SUBind,ADDind,ORAind,STAind,ANDind: begin
                                PC<=PC;
                                RDR<=RDR+16'b1;
                                PB<=BUSDAT;
                            end
```

Diagrama de temporización corregido:

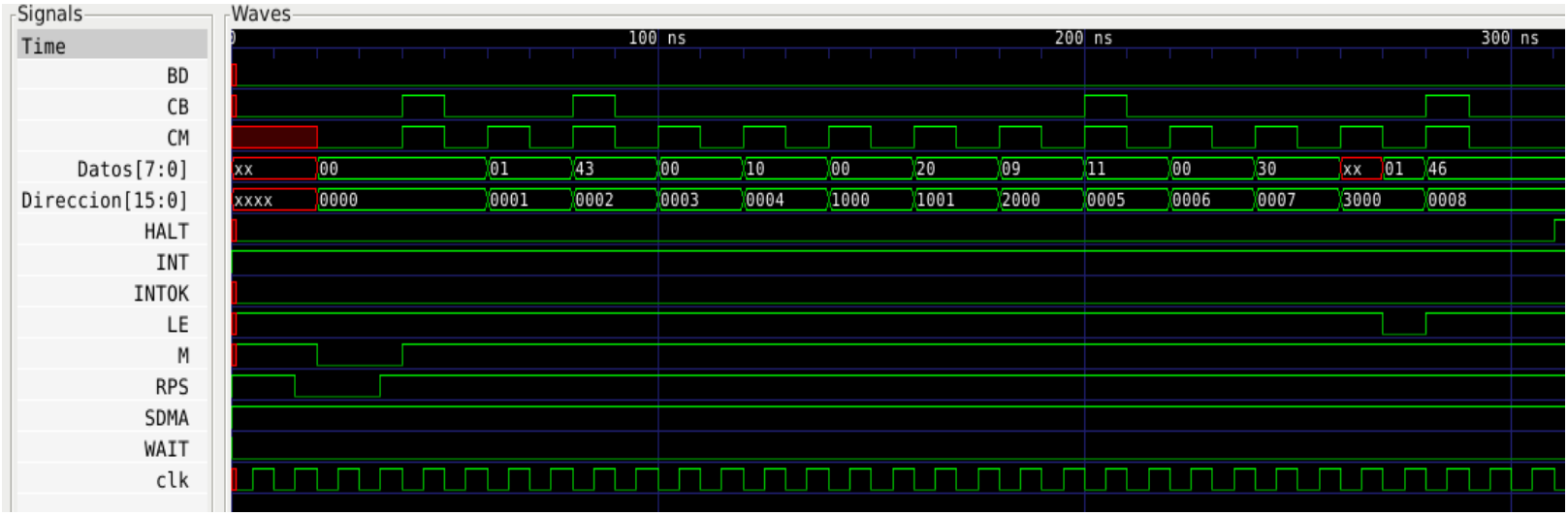

Figura 16. ANDind correcto

ORAind (OR Accumulator indirecto)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 ORA ($1000)
$0005 STA $3000
$0008 HLT

$1000 $00
$1001 $20

$2000 $09
```

Diagrama de temporización:

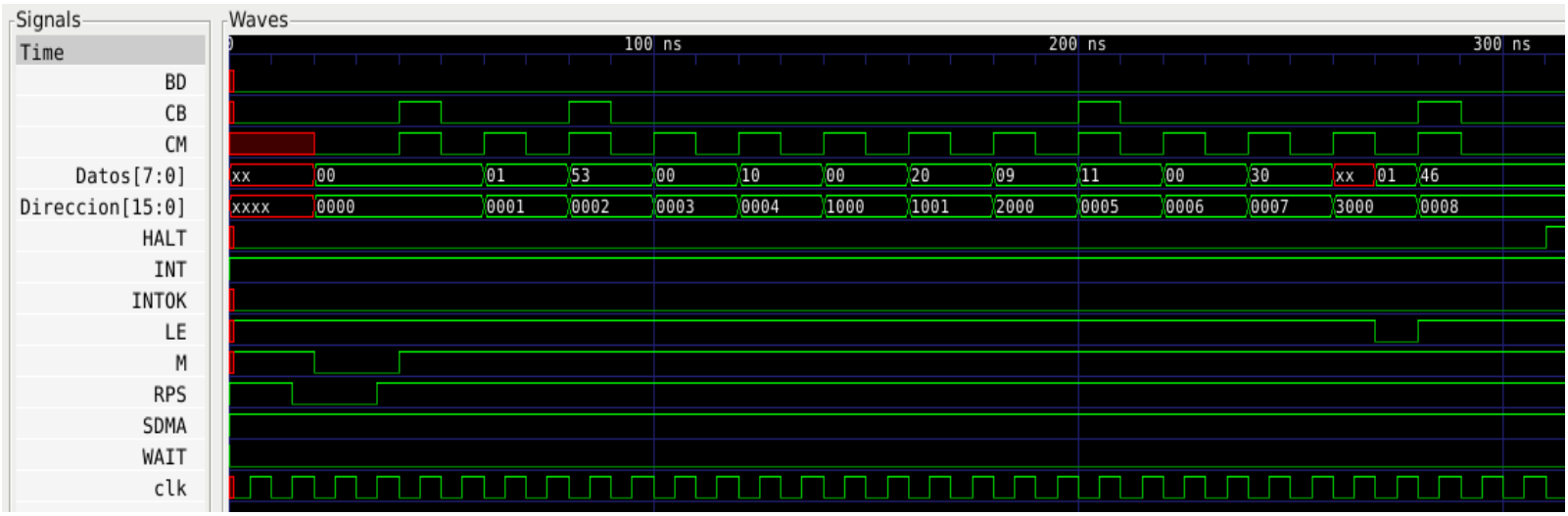

Figura 17. ORAind incorrecto

Código incorrecto en Verilog: (en el módulo acumulador)

```
ORAind: A<=A&BUSDAT;
```

Código correcto en Verilog:

```
ORAind: A<=A|BUSDAT;
```

Diagrama de temporización corregido:

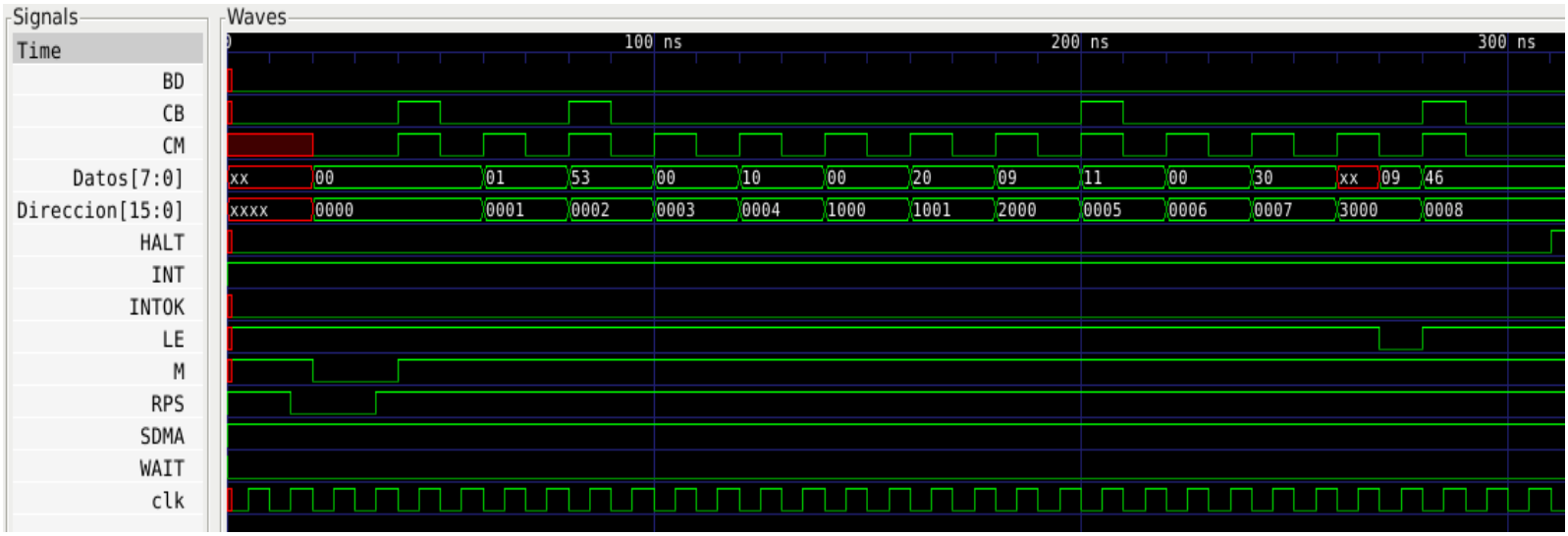

Figura 18. ORAind correcto

JSRind (Jump SubRoutine indirecto)

Programa en ensamblador:

```
$0000 CLA
$0001 TAP
$0002 JSR ($1000)

$1000 $00
$1001 $20

$2000 HLT
```

Diagrama de temporización:

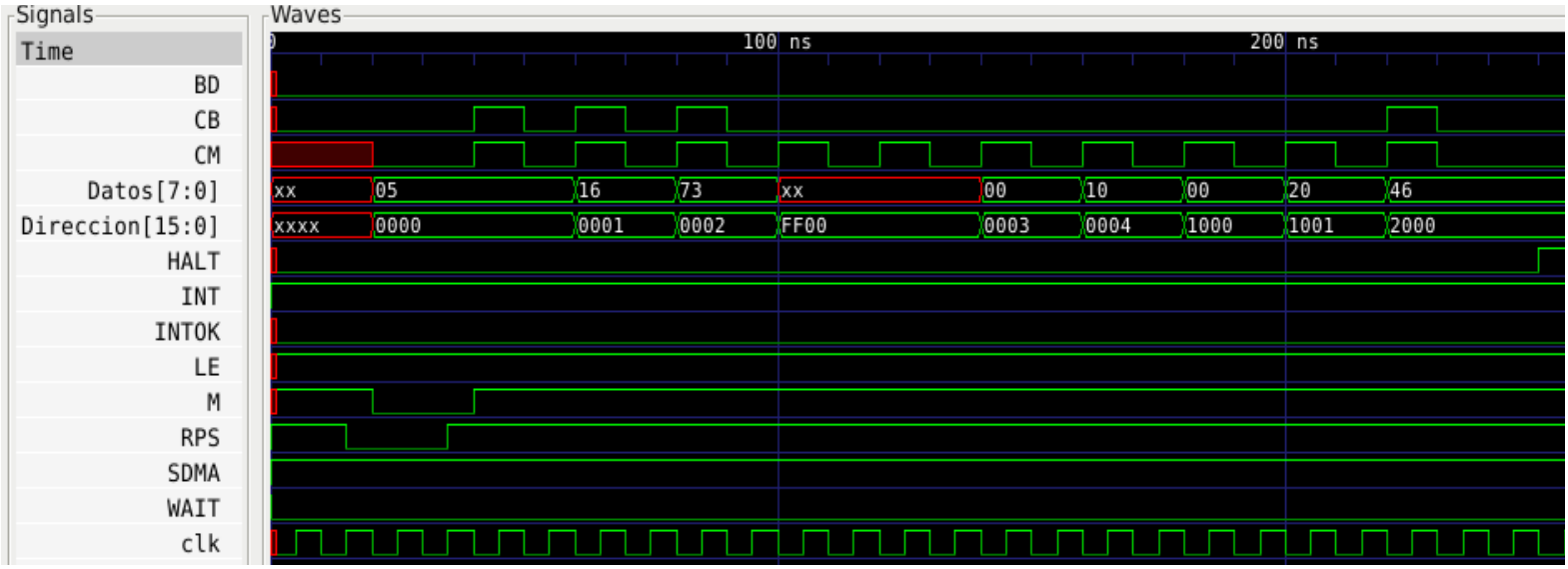

Figura 19. JSRind incorrecto

Código incorrecto en Verilog: (en el módulo reg_LE)

```
Estado_3: case(RI)
            PHA,PHS: LE<=1'b0;
            JSR: LE<=1'b0;
            default: LE<=1'b1;
          endcase
 Estado_5: case(RI)
            JSR,OUT:LE<=1'b0;
            default: LE<=1'b1;
          endcase
```

Código correcto en Verilog:

```
Estado_3: case(RI)
            PHA,PHS: LE<=1'b0;

            //se agrego la instruccion JSRind
            JSR,JSRind: LE<=1'b0;
            default: LE<=1'b1;
          endcase
 Estado_5: case(RI)

            //se agrego la instruccion JSRind
            JSR,OUT,JSRind:LE<=1'b0;
            default: LE<=1'b1;
          endcase
```

Diagrama de temporización corregido:

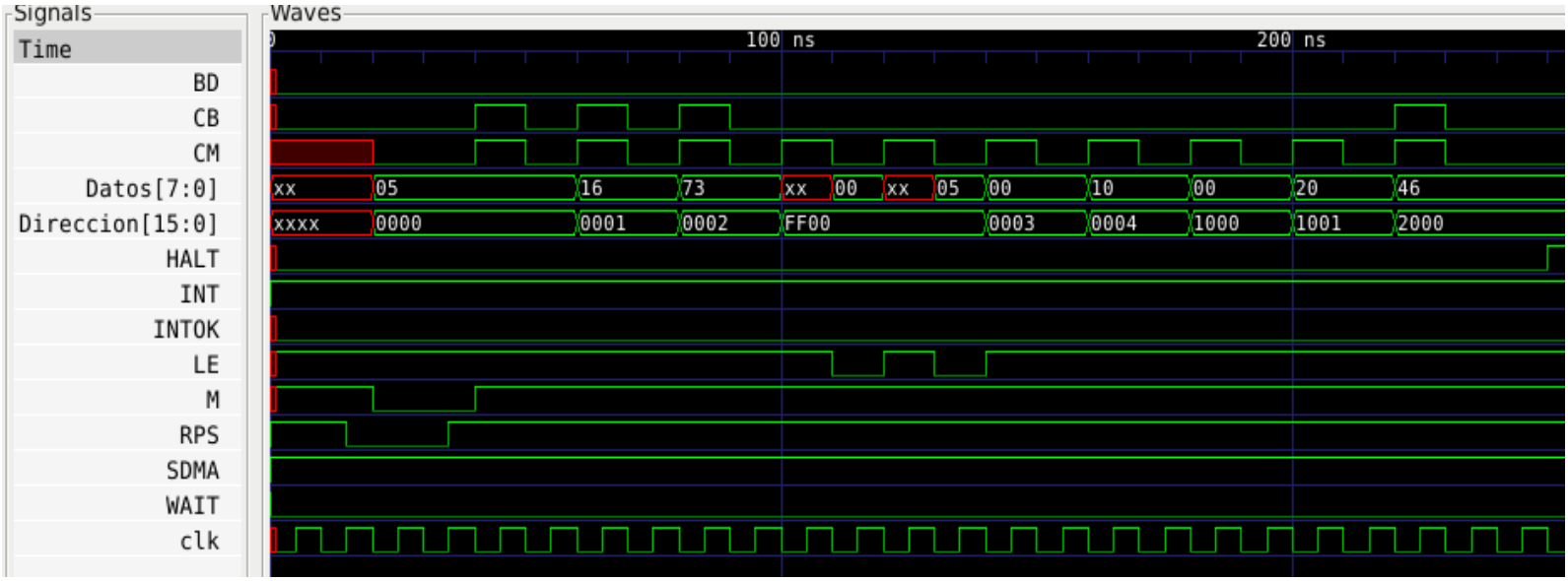

Figura 20. JSRind correcto

JMPind (JuMP indirecto)

Programa en ensamblador:

```
$0000 JMP ($1000)

$1000 $00
$1001 $20

$2000 HLT
```

Diagrama de temporización:

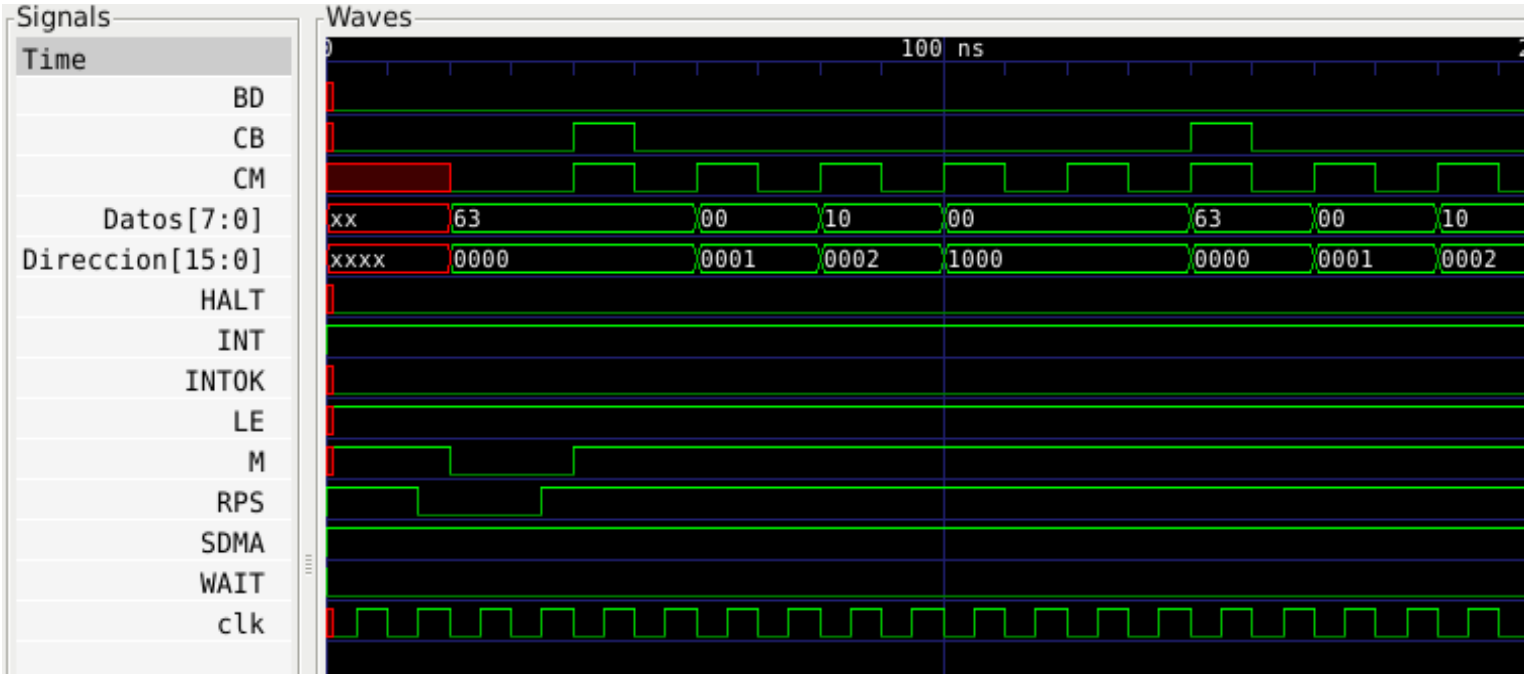

Figura 21. JMPind incorrecto

Código incorrecto en Verilog: (en el módulo reg_PC)

```
Estado_8: begin
            case(RI)
                STA,LDA,AND,SUB,ORA,ADD: begin
                        PC<=PC;
                        RDR<=PC;
                    end
                    //se agrego la instruccion ANDind a este caso
                LDAind,SUBind,ADDind,ORAind,STAind,ANDind: begin
                        PC<=PC;
                        RDR<=RDR+16'b1;
                        PB<=BUSDAT;
```

```
end
```

Código correcto en Verilog:

```
Estado_8: begin
            case(RI)
                STA,LDA,AND,SUB,ORA,ADD: begin
                    PC<=PC;
                    RDR<=PC;
                end
                //se agrego la instruccion ANDind a este caso
                //se agrego la instruccion JMPind a este caso
                LDAind,SUBind,ADDind,ORAind,STAind,ANDind,JMPind: begin
                    PC<=PC;
                    RDR<=RDR+16'b1;
                    PB<=BUSDAT;
                end
```

Diagrama de temporización corregido:

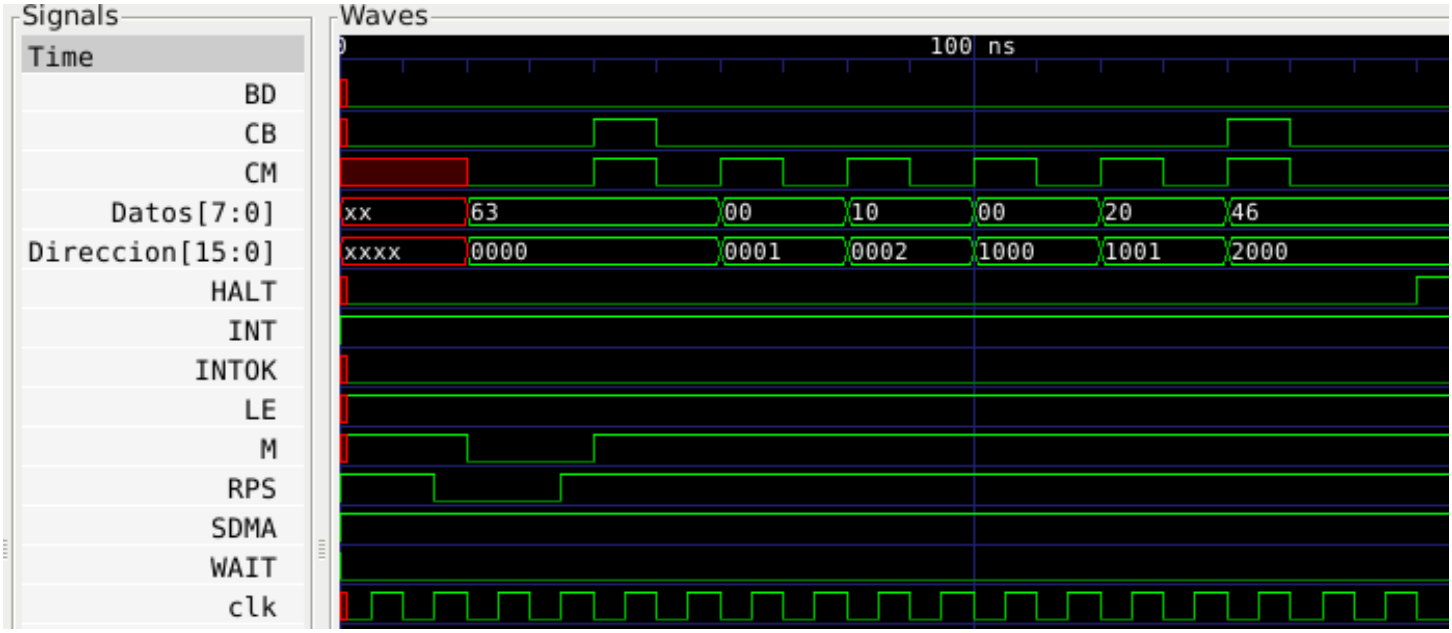

Figura 22. JMPind correcto

II PARTE:

INSTRUCCIONES CORRECTAS

LDAinm (LoaD Accumulator inmediato)

Programa en ensamblador:

```
$0000 LDA #$06
$0002 HLT
```

Diagrama de temporización:

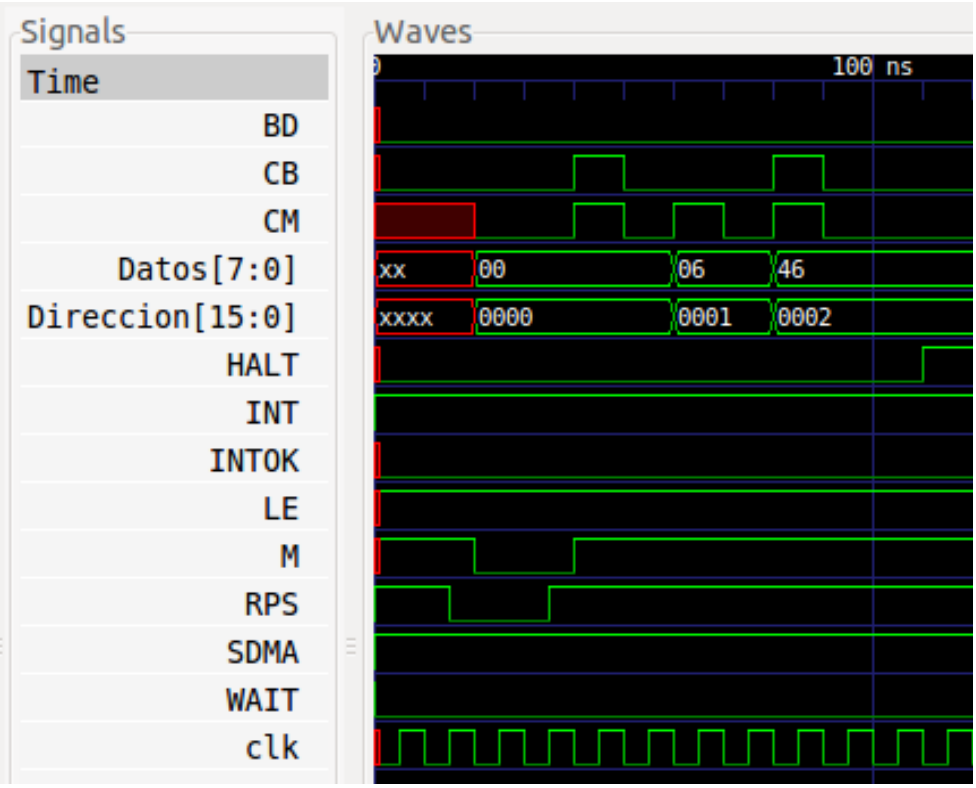

Figura 23. LDAinm correcto

ADDinm (ADD inmediato)

Programa en ensamblador:

```
$0000 LDA #$04
$0002 ADD #$06
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

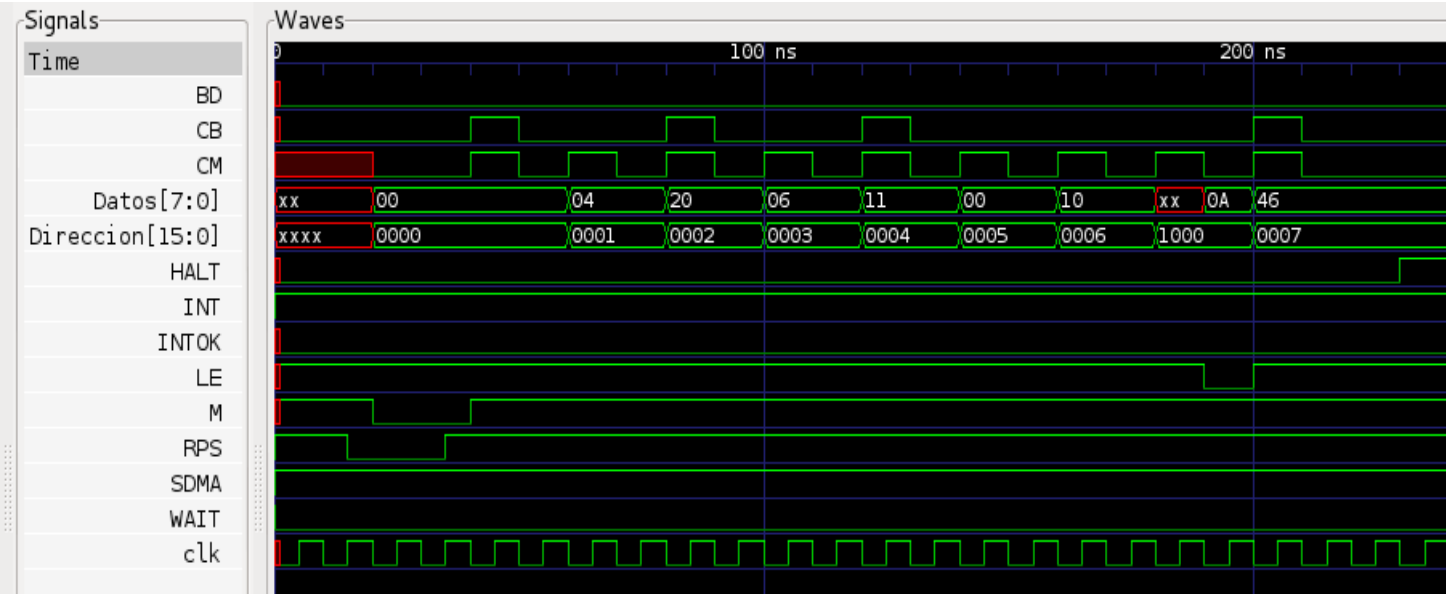

Figura 24. ADDinm correcto

ANDinm (AND inmediato)

Programa en ensamblador:

```
$0000 LDA #$06
$0002 AND #$02
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

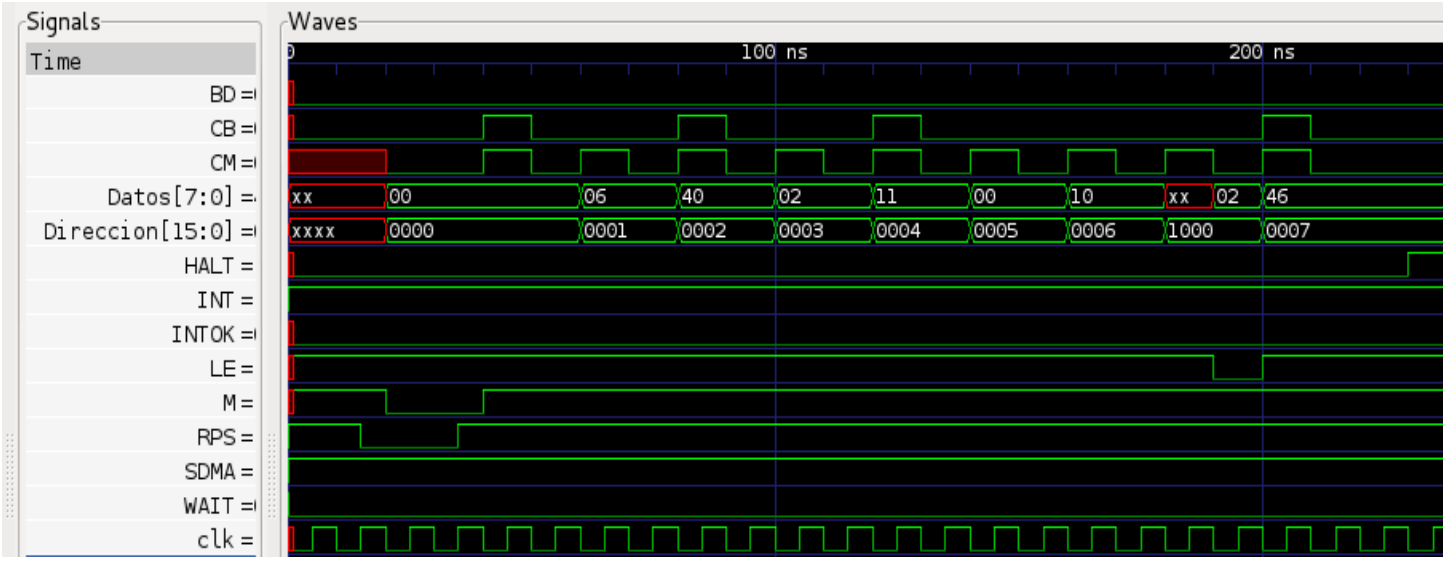

Figura 25. ANDinm correcto

ORAinm (ORA inmediato)

Programa en ensamblador:

```
$0000 LDA #$06
$0002 ORA #$02
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

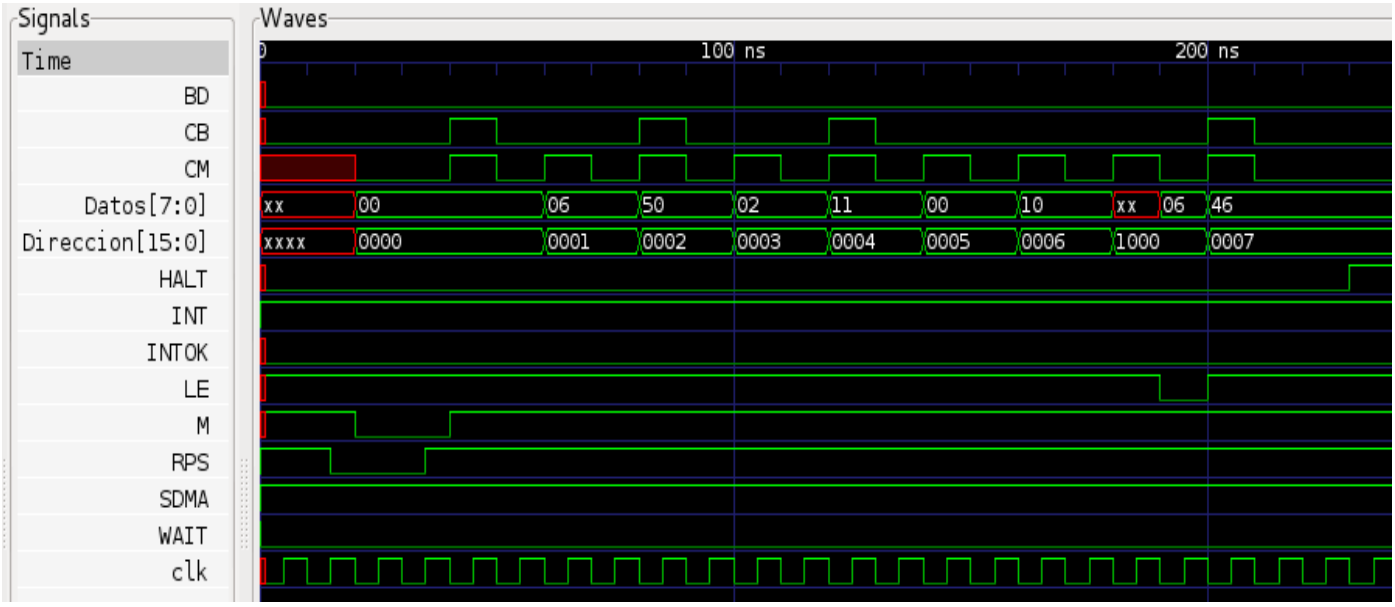

Figura 26. ORAinm correcto

LDA (LoaD Accumulator)

Programa en ensamblador:

(En la posición 1000 se escribió previamente un $FF)

```
$0000 LDA $1000
$0003 HLT
```

Diagrama de temporización:

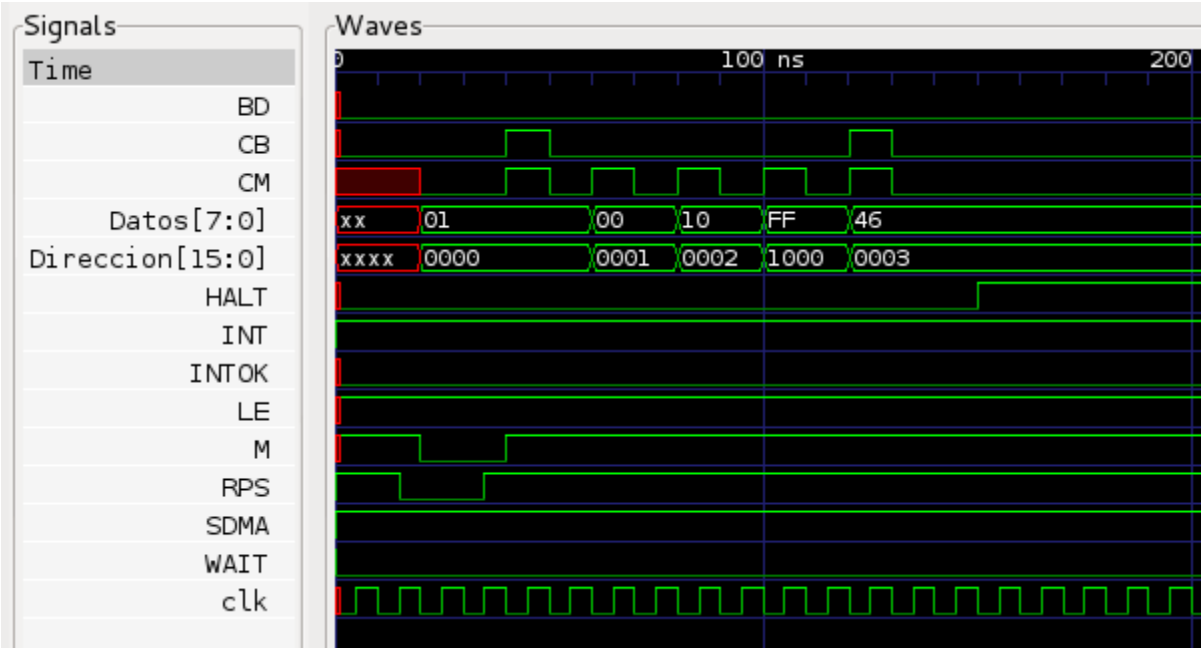

Figura 27. LDA correcto

ADD (ADD)

Programa en ensamblador:

(Previamente se escribió en la posición $2000 de la memoria un $02)

```
$0000 LDA #$06
$0002 ADD $2000
$0005 STA $1000
$0008 HLT
```

Diagrama de temporización:

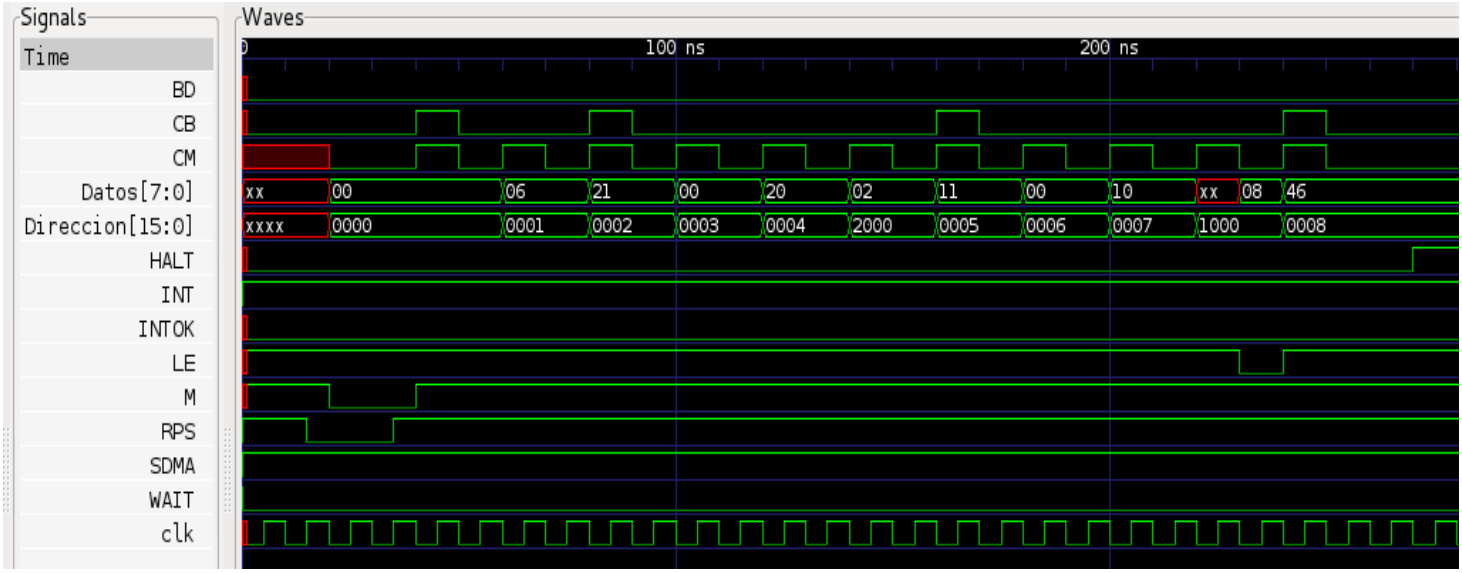

Figura 28. ADD correcto

SUB (SUBtract)

Programa en ensamblador:

(Previamente se escribió en la posición $2000 de la memoria un $02)

```
$0000 LDA #$06
$0002 SUB $2000
$0005 STA $1000
$0008 HLT
```

Diagrama de temporización:

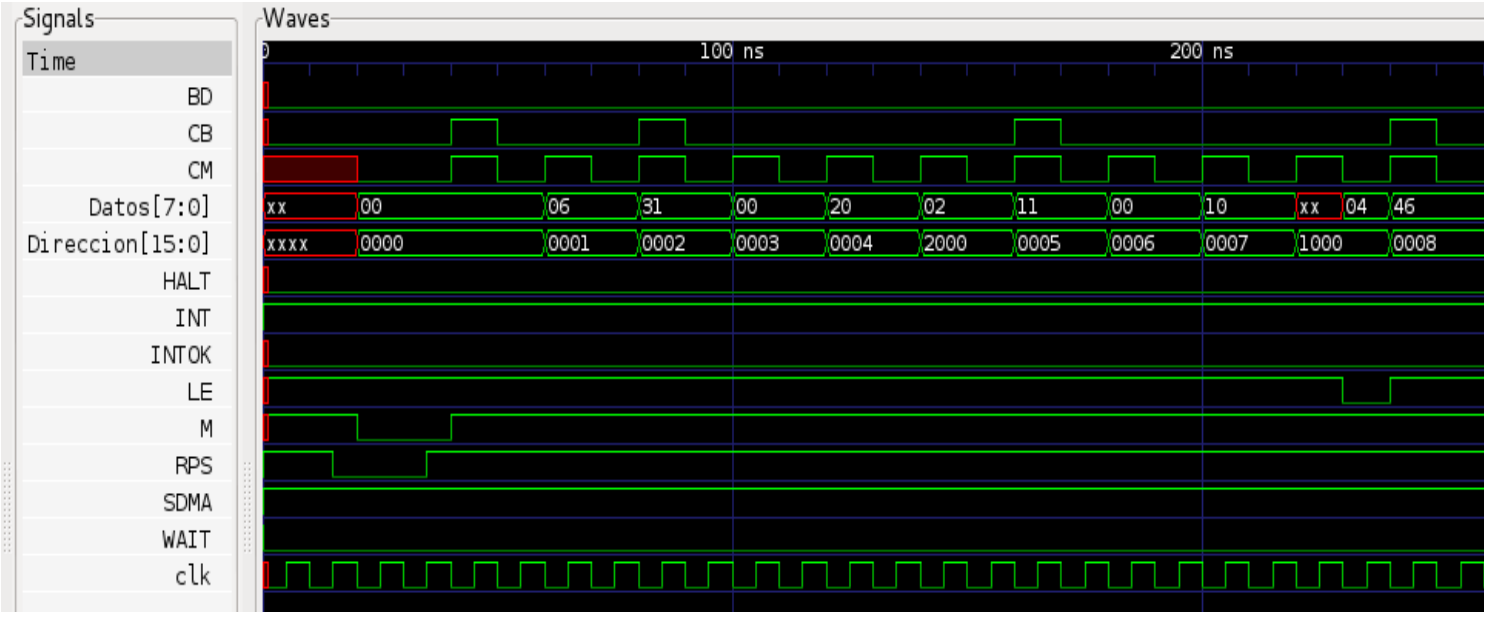

Figura 29. SUB correcto

AND (AND)

Programa en ensamblador:

(Previamente se escribió en la posición $2000 de la memoria un $05)

```
$0000 LDA #$06
$0002 AND $2000
$0005 STA $1000
$0008 HLT
```

Diagrama de temporización:

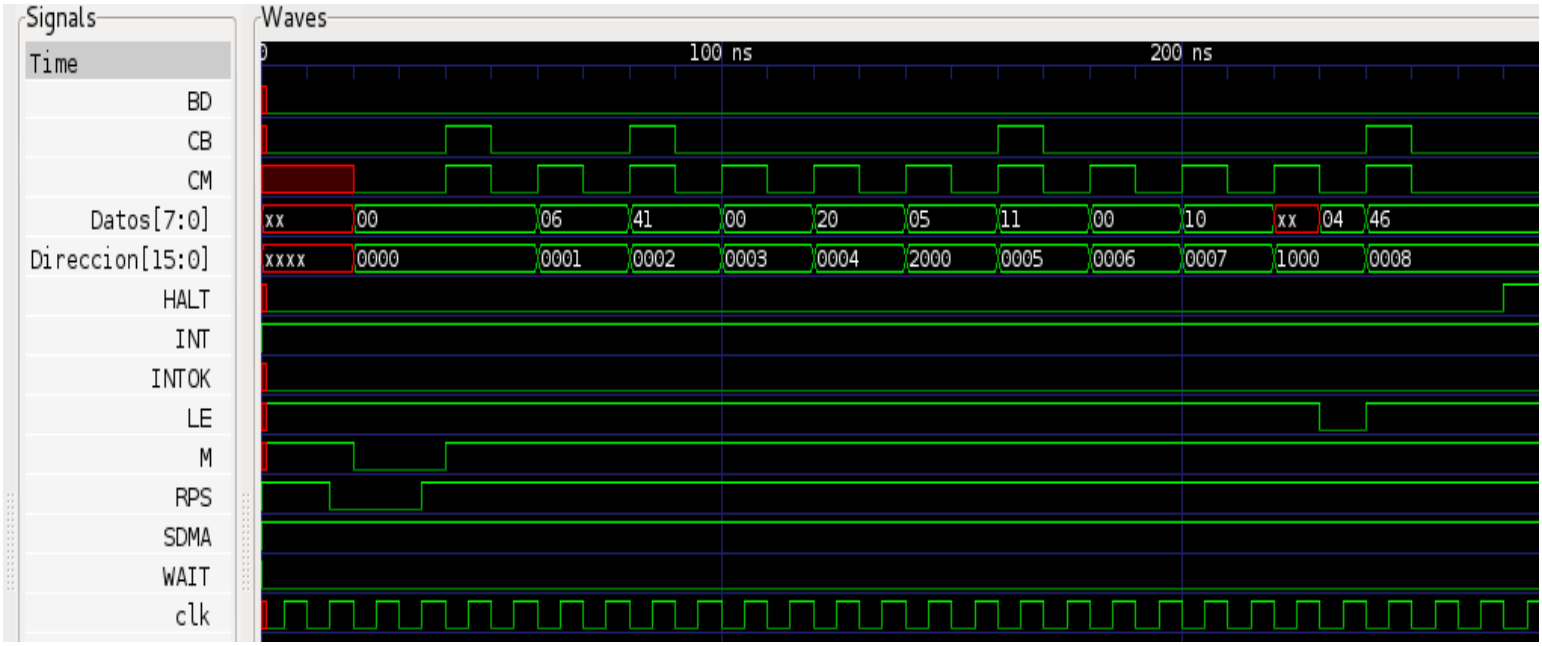

Figura 30. AND correcto

ORA (OR Accumulator)

Programa en ensamblador:

(Previamente se escribió en la posición $2000 de la memoria un $05)

```
$0000 LDA #$06
$0002 ORA $2000
$0005 STA $1000
$0008 HLT
```

Diagrama de temporización:

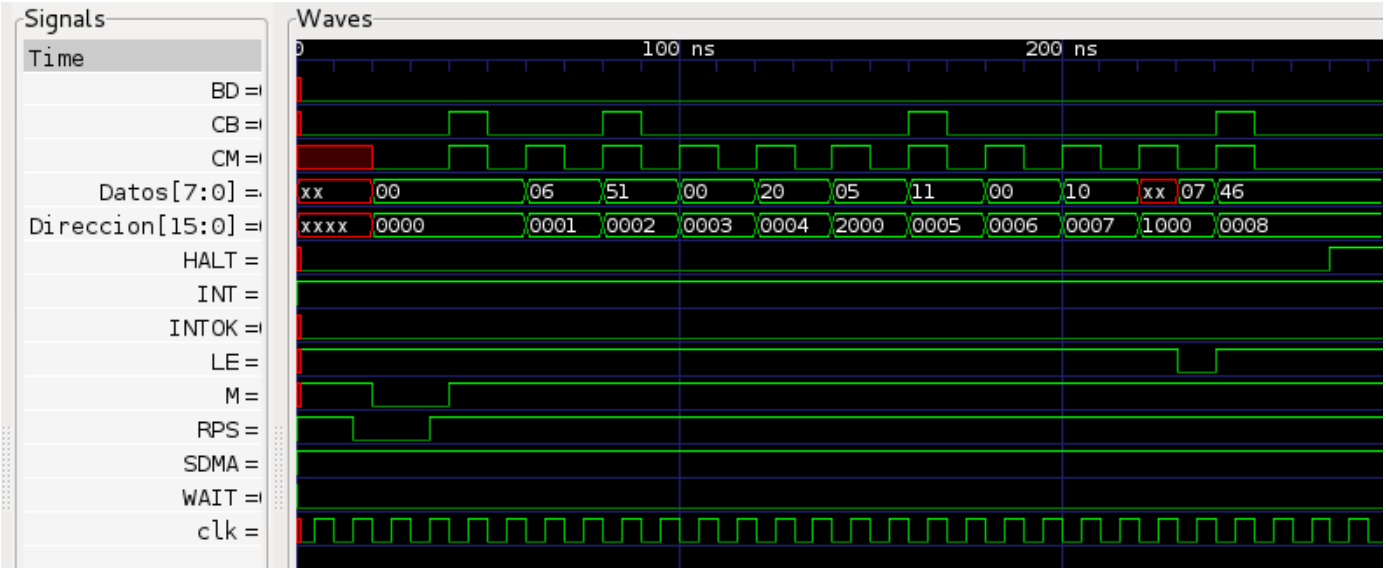

Figura 31. ORA correcto

JMP (JuMP)

Programa en ensamblador:

(Previamente se escribió en la posición $2000 de la memoria un $05)

```
$0000 LDA #$06
$0002 ORA $2000
$0005 JMP $0004
$0008 STA $1000
$000B HLT
```

Diagrama de temporización:

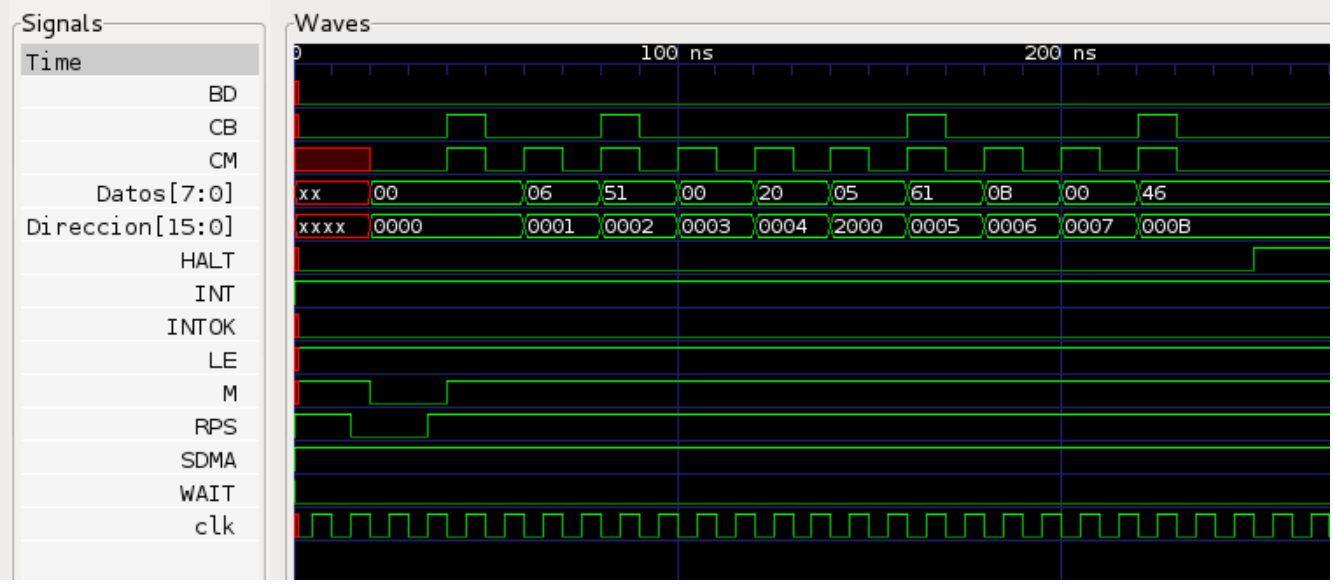

Figura 32. JMP correcto

TAP-PHA(Transfer A to P -PusH Accumulator)

(Las dos instrucciones fueron evaluadas simultáneamente)

Programa en ensamblador:

```
$0000 CLA
$0001 TAP
$0002 PHA
$0003 HLT
```

Diagrama de temporización:

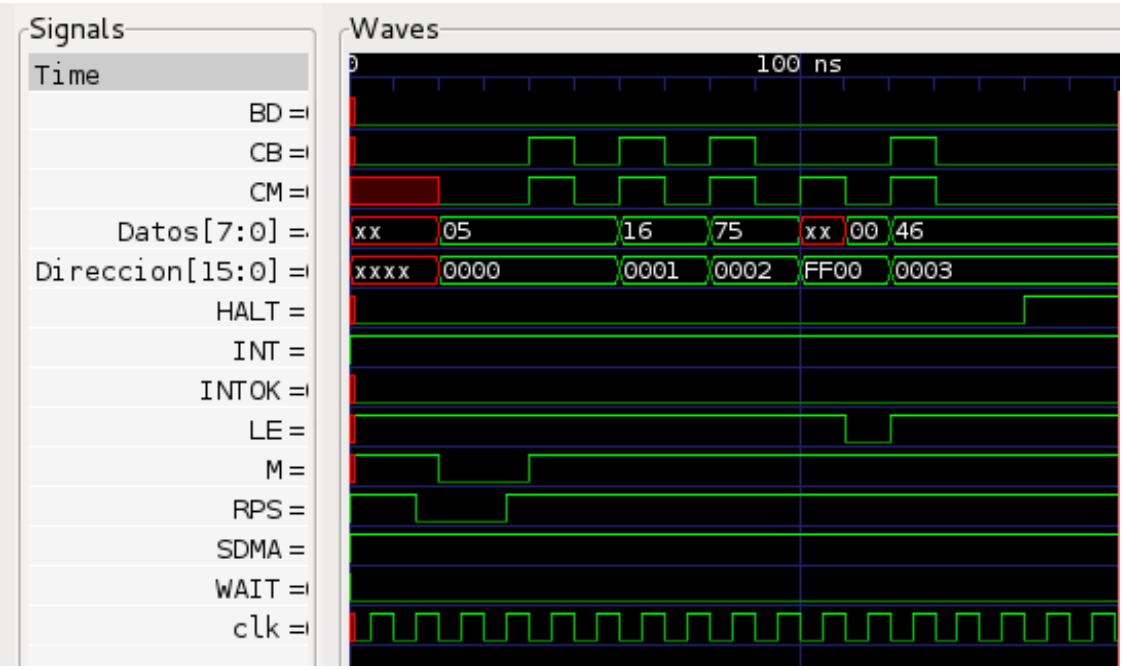

Figura 33. TAP-PHA correctas

JSR (Jump SubRoutine)

Programa en ensamblador:

```
$0000 CLA
$0001 TAP
$0002 JSR $2000
$2000 HLT
```

Diagrama de temporización:

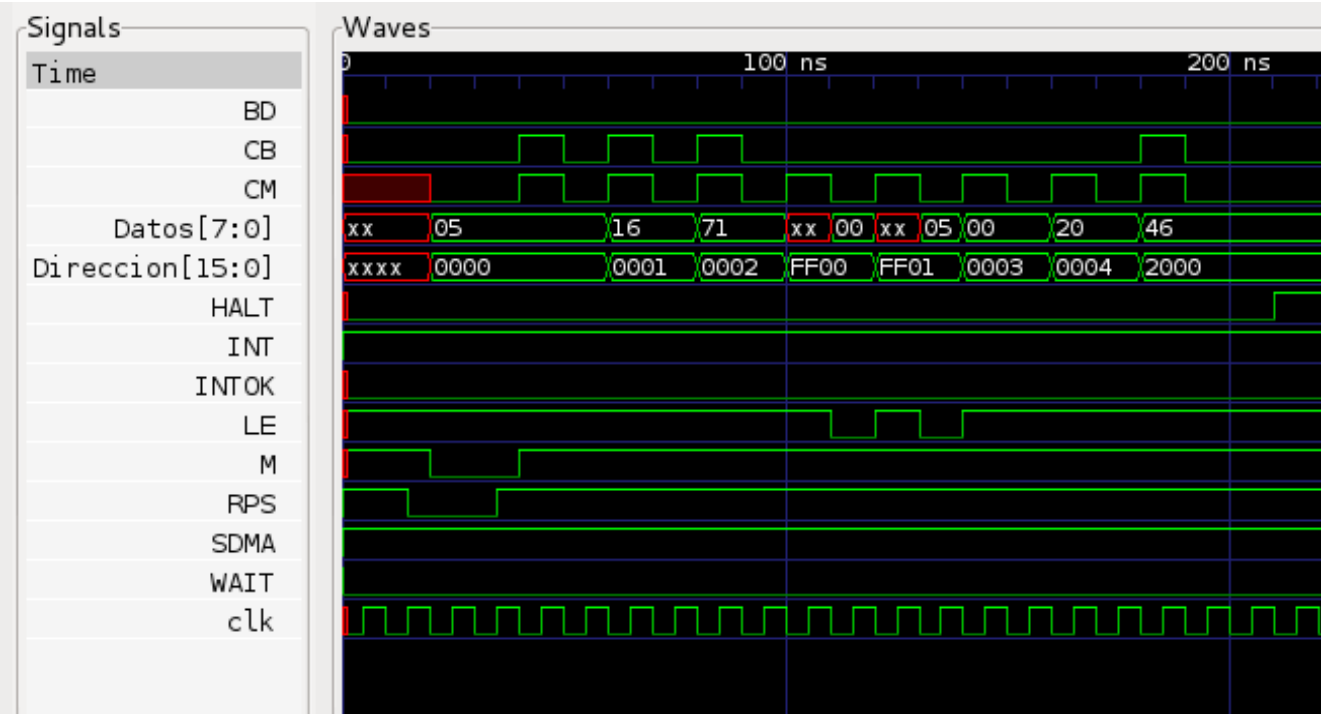

Figura 34. JSR correcto

RTS (ReTurn from Subroutine)

Programa en ensamblador:

```
$0000 LDA #$02
$0002 TAP
$0003 RTS
$2000 HLT
$FF00 $20
$FF01 $00
```

Diagrama de temporización:

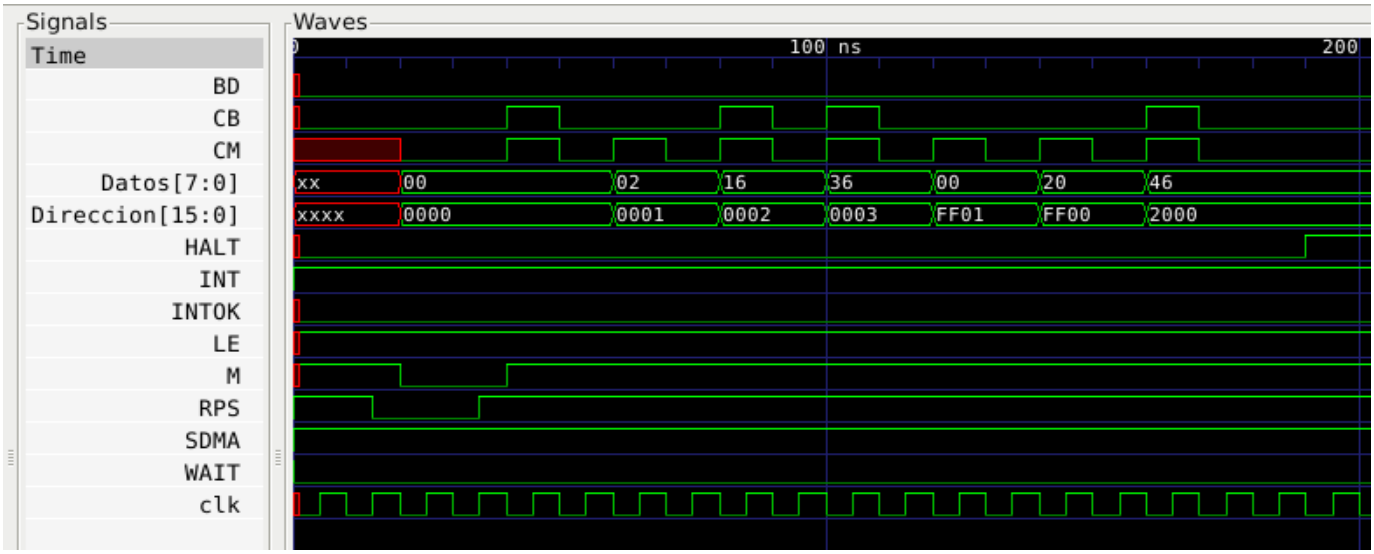

Figura 35. RTS correcto

PLS (PuLl Status)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 TAP
$0003 PLS
$0004 HLT
$FF00 $2F
```

Diagrama de temporización:

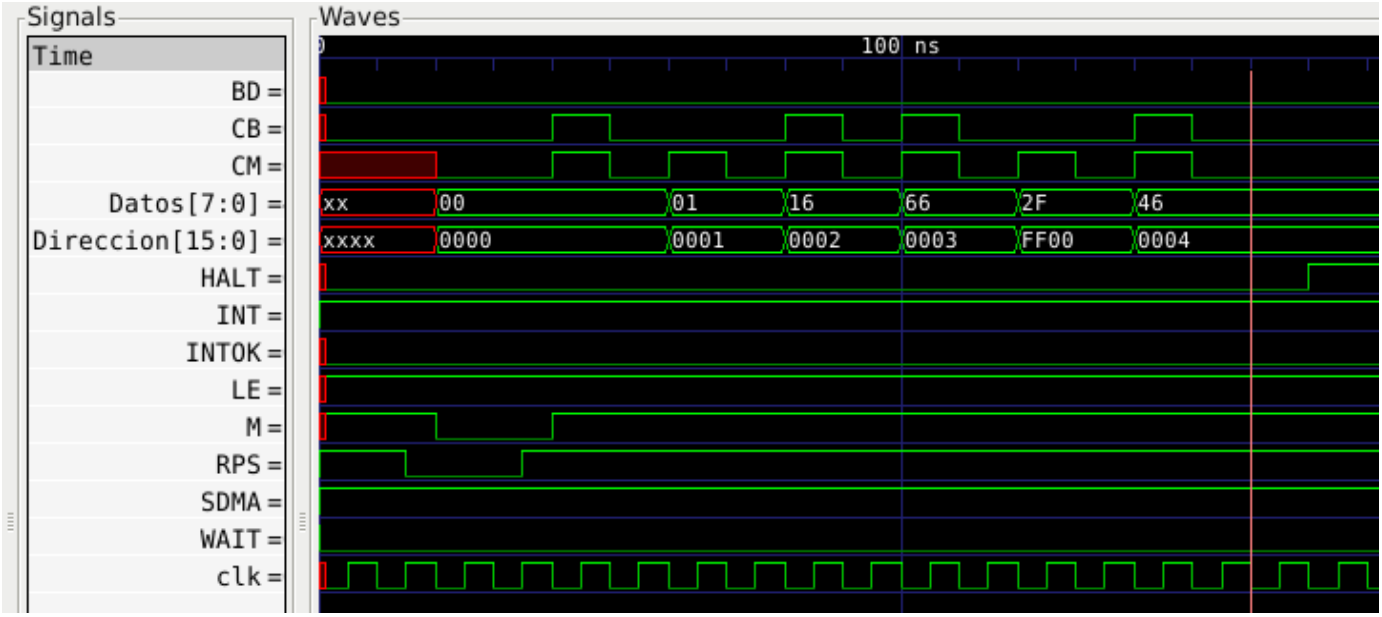

Figura 36. PLS correcto

PHS (PusH Status)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 TAP
$0003 PLS
$0004 PHS
$0005 HLT
$FF00 $2F
```

Diagrama de temporización:

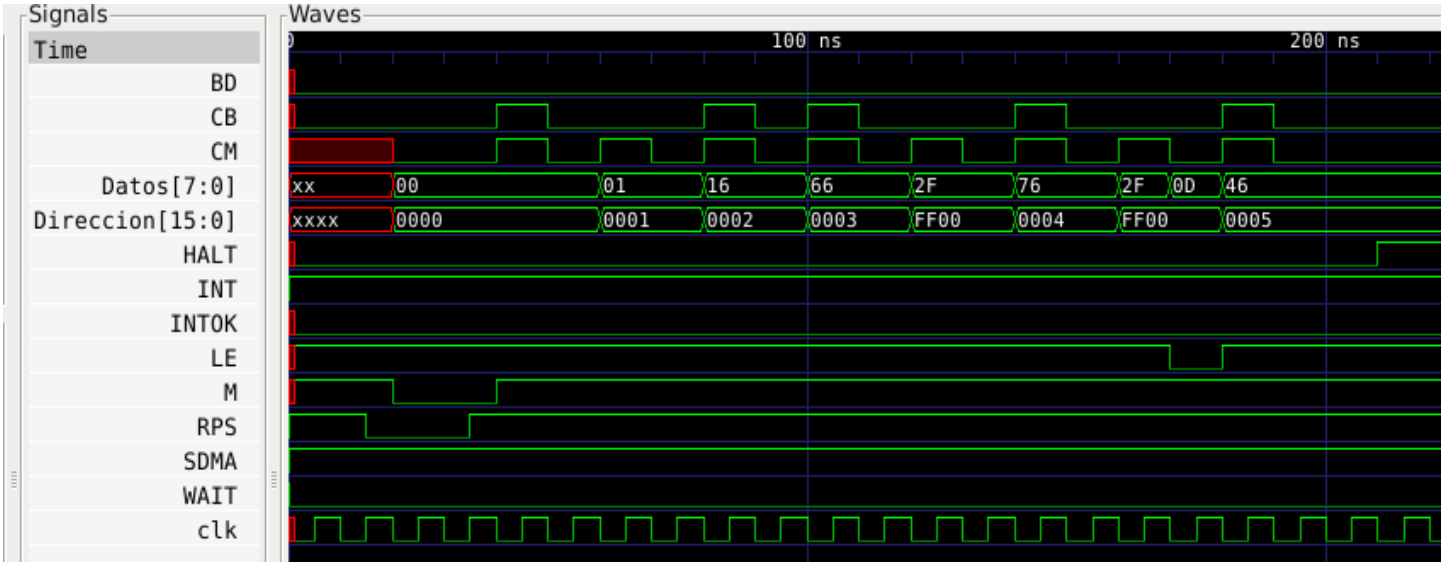

Figura 37. PHS correcto

BEQ (Branch if EQual)

Programa en ensamblador:

Con el fin de saltar:

(Se guardó previamente un cero en la posición $3000)

```
$0000 CLA
$0001 LDA $3000
$0004 BEQ $0004
$0006 INA
$0007 HLT
```

Con el fin de no saltar:

(Se guardó previamente un uno en la posición $3000)

```
$0000 CLA
$0001 LDA $3000
$0004 BEQ $0004
$0006 INA
$0007 HLT
```

Diagrama de temporización:

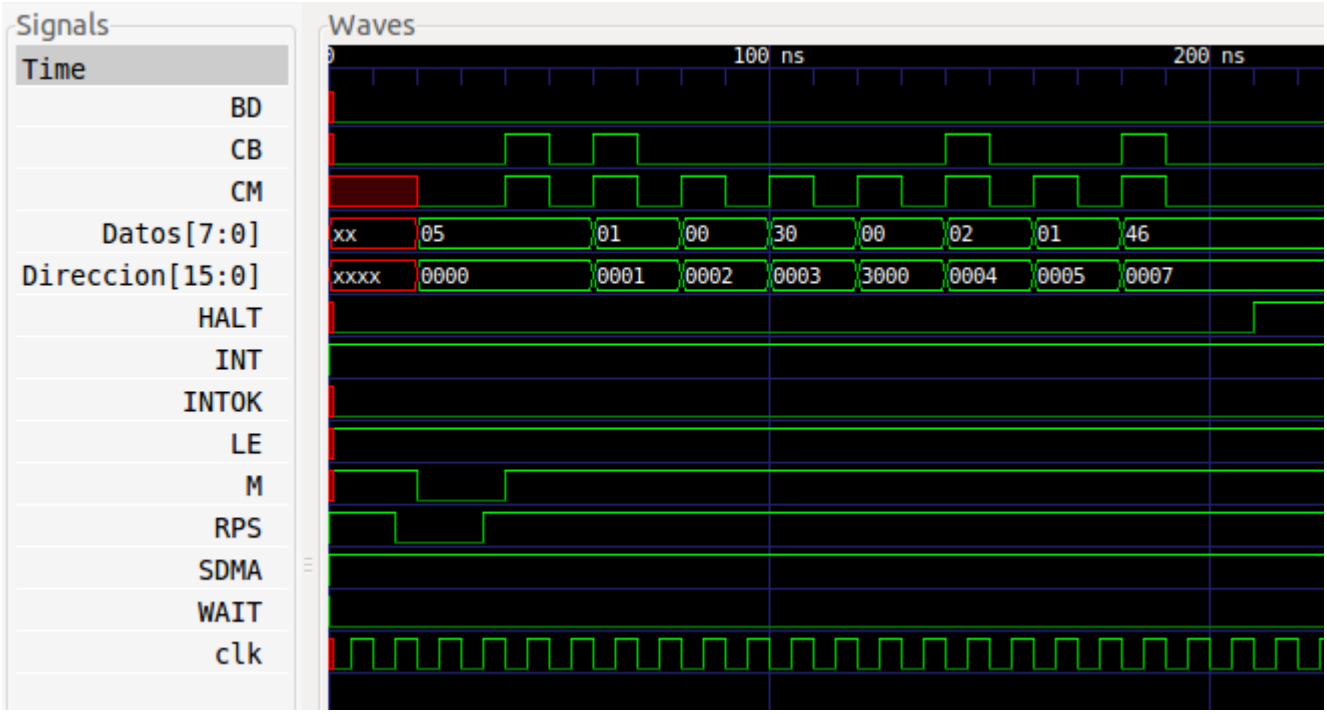

Figura 38. BEQ cuando salta, correcto

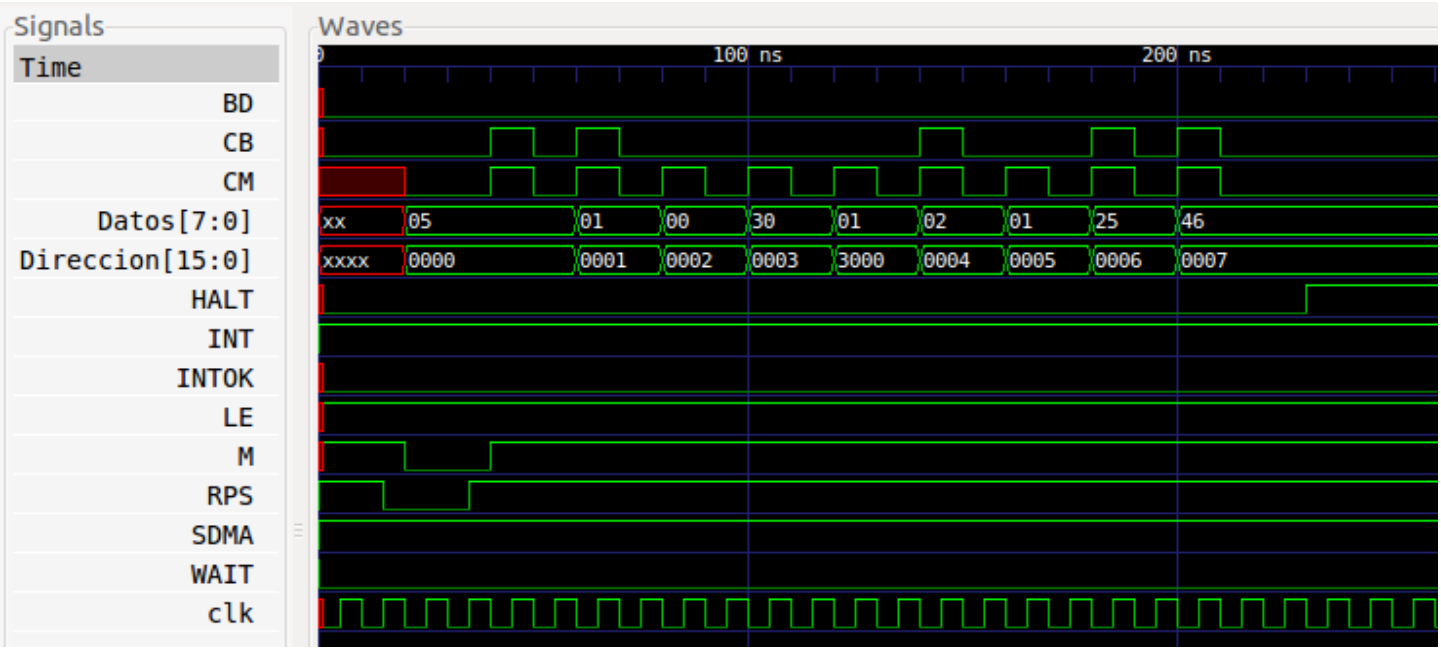

Figura 39. BEQ cuando no salta, correcto

BNE (Branch if Not Equal)

Programa en ensamblador:

Con el fin de saltar:

(Se guardó previamente un uno en la posición $3000)

```
$0000 CLA
$0001 LDA $3000
$0004 BNE $0004
$0006 INA
$0007 HLT
```

Con el fin de no saltar:

(Se guardó previamente un cero en la posición $3000)

```
$0000 CLA
$0001 LDA $3000
$0004 BNE $0004
$0006 INA
$0007 HLT
```

Diagrama de temporización:

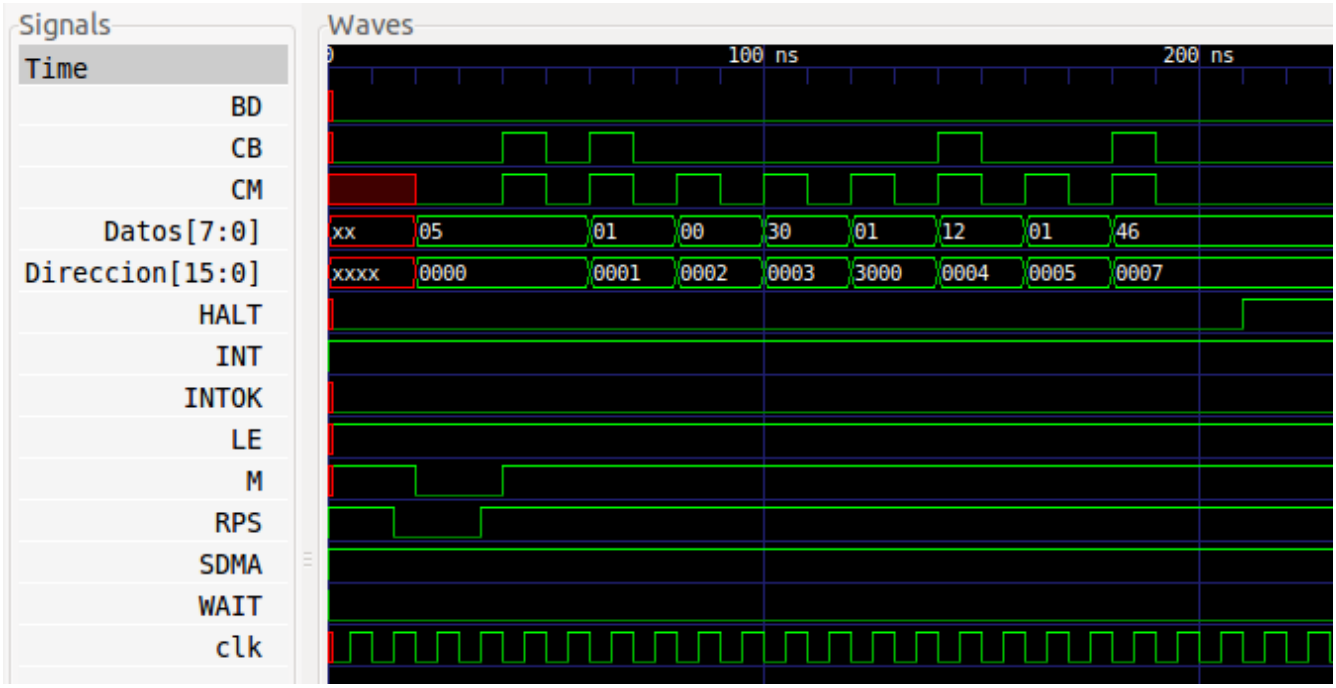

Figura 40. BNE cuando salta, correcto

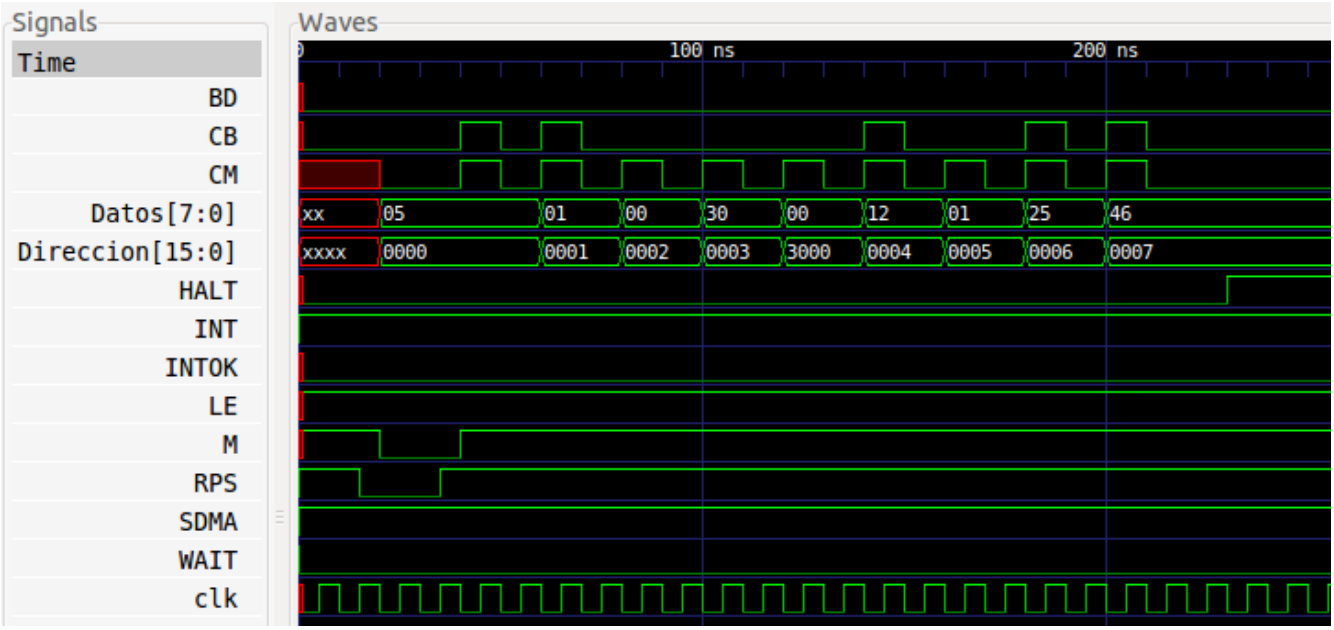

Figura 41. BNE cuando no salta, correcto

CLC (CLear Carry flag)

Programa en ensamblador:

```
$0000 LDA #$00
$0002 TAP
$0003 CLC
$0004 PHS
$0005 HLT
```

Diagrama de temporización:

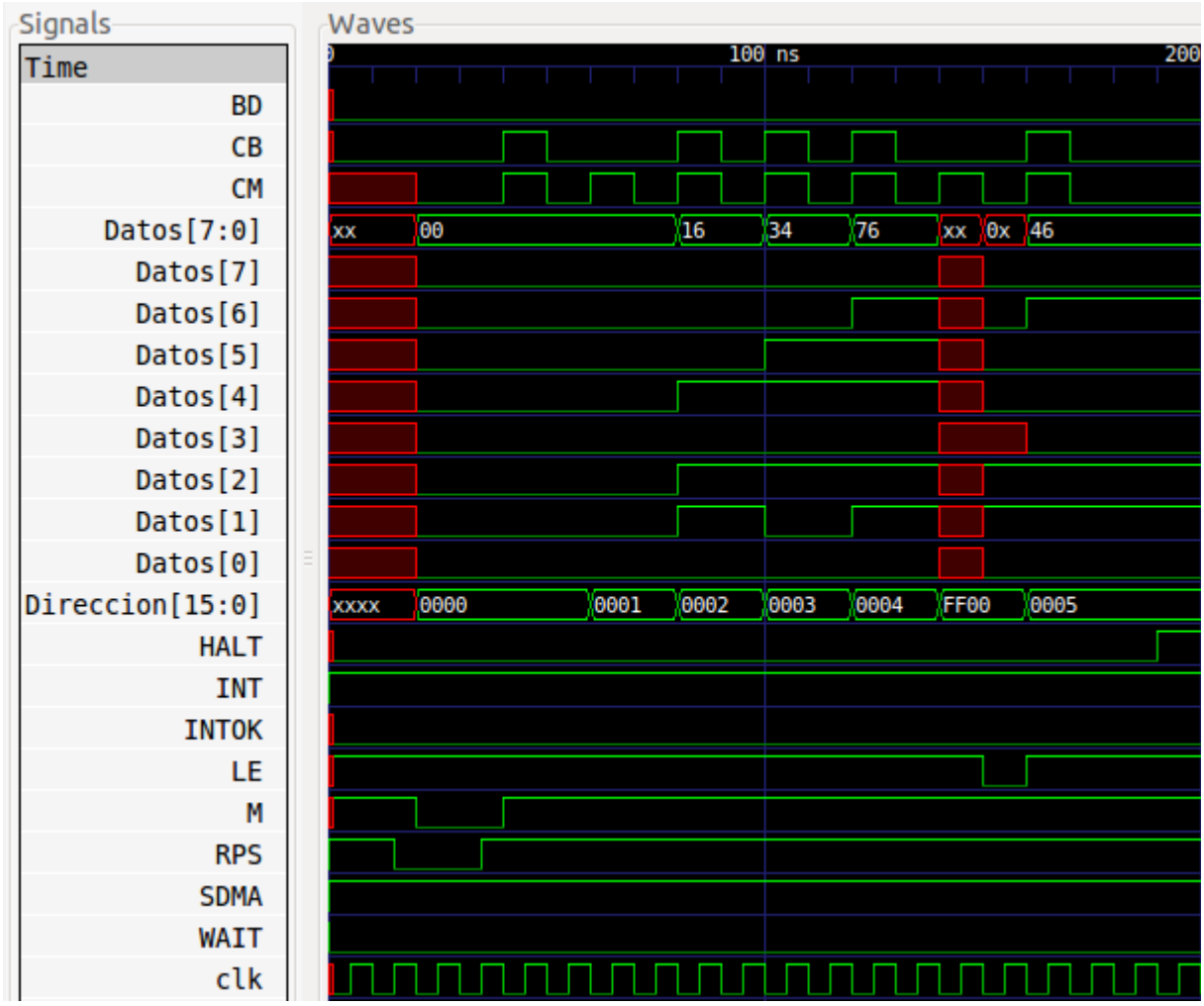

Figura 42. CLCcorrecto

BCS (Branch if Carry Set)

Programa en ensamblador:

Con el fin de saltar:

```
$0000 CLC
$0001 LDA #$FF
$0003 ADD #$01
$0005 BCS $0005
$0007 STA $1000
$000A HLT
```

Con el fin de no saltar:

```
$0000 CLC
$0001 LDA #$FE
$0003 ADD #$01
$0005 BCS $0005
$0007 STA $1000
$000A HLT
```

Diagrama de temporización:

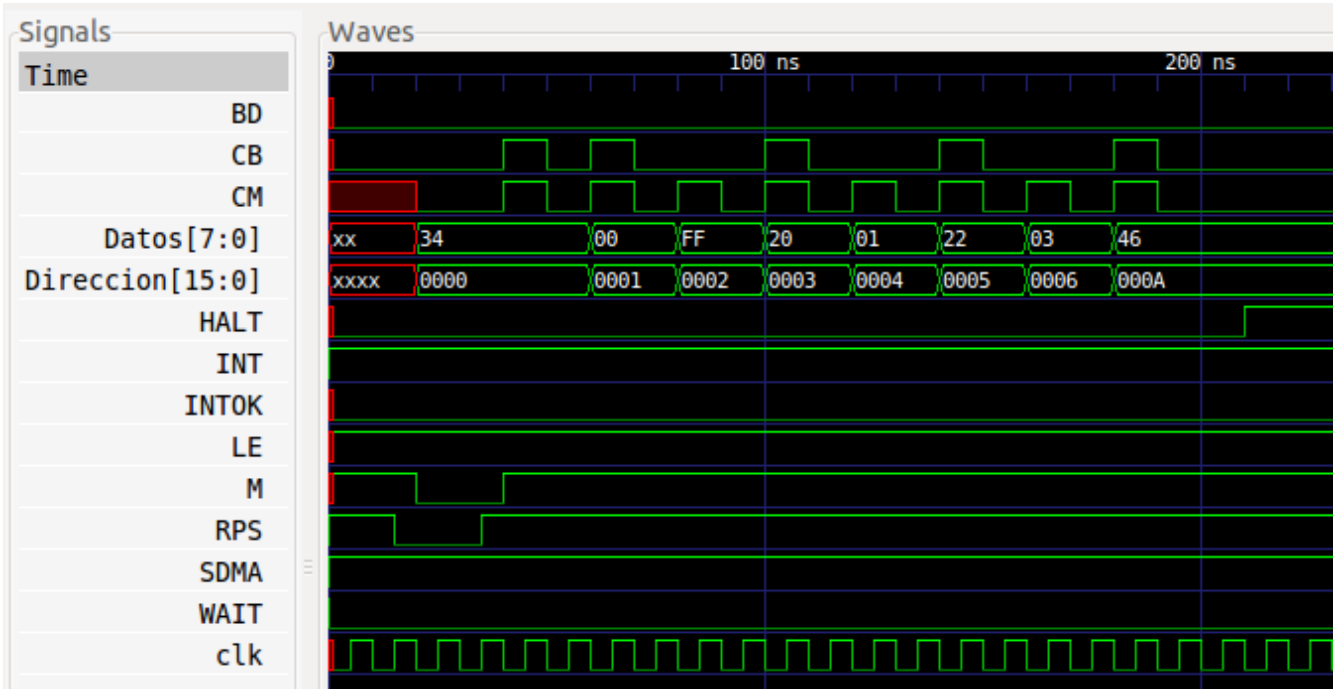

Figura 43. BCS cuando salta, correcto

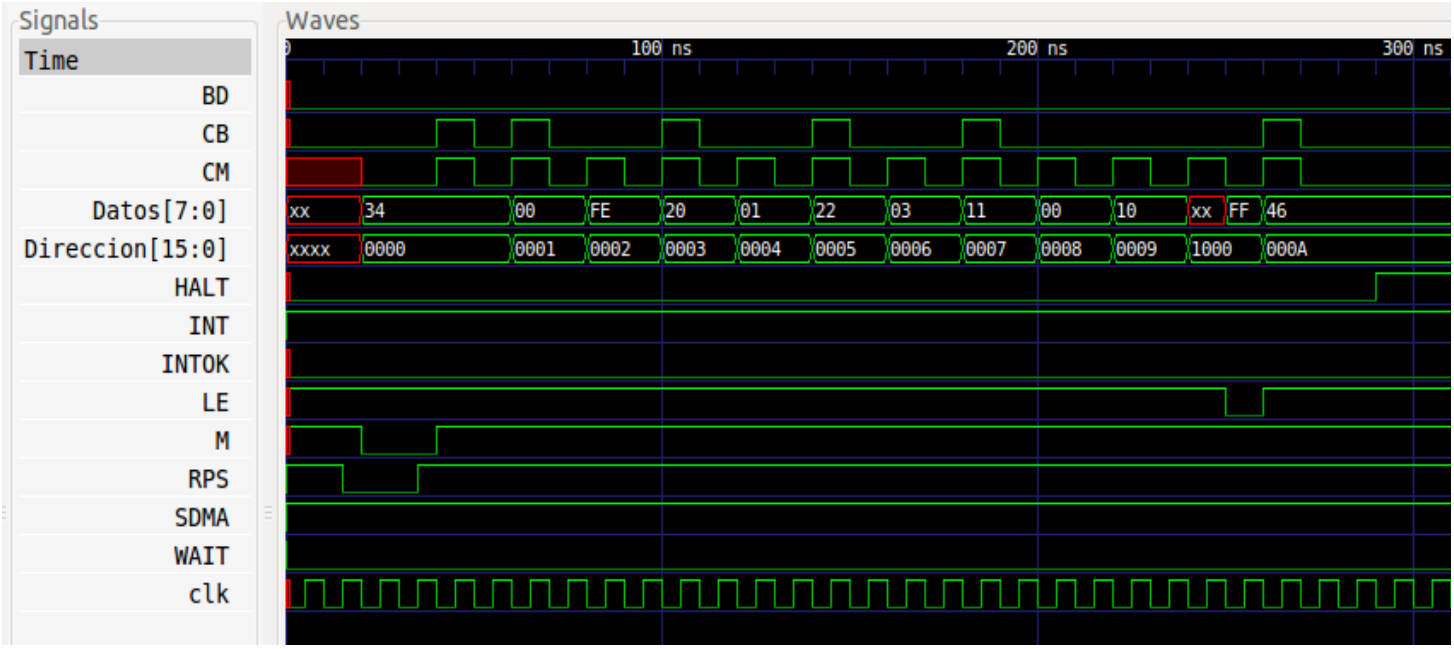

Figura 44. BCS cuando no salta, correcto

BPL (Branch if PLus)

Programa en ensamblador:

Con el fin de saltar:

```
$0000 LDA #$00
$0002 SUB #$01
$0004 BPL $0004
$0006 STA $1000
$0009 HLT
```

Con el fin de no saltar:

```
$0000 LDA #$01
$0002 SUB #$01
$0004 BPL $0004
$0006 STA $1000
$0009 HLT
```

Diagrama de temporización:

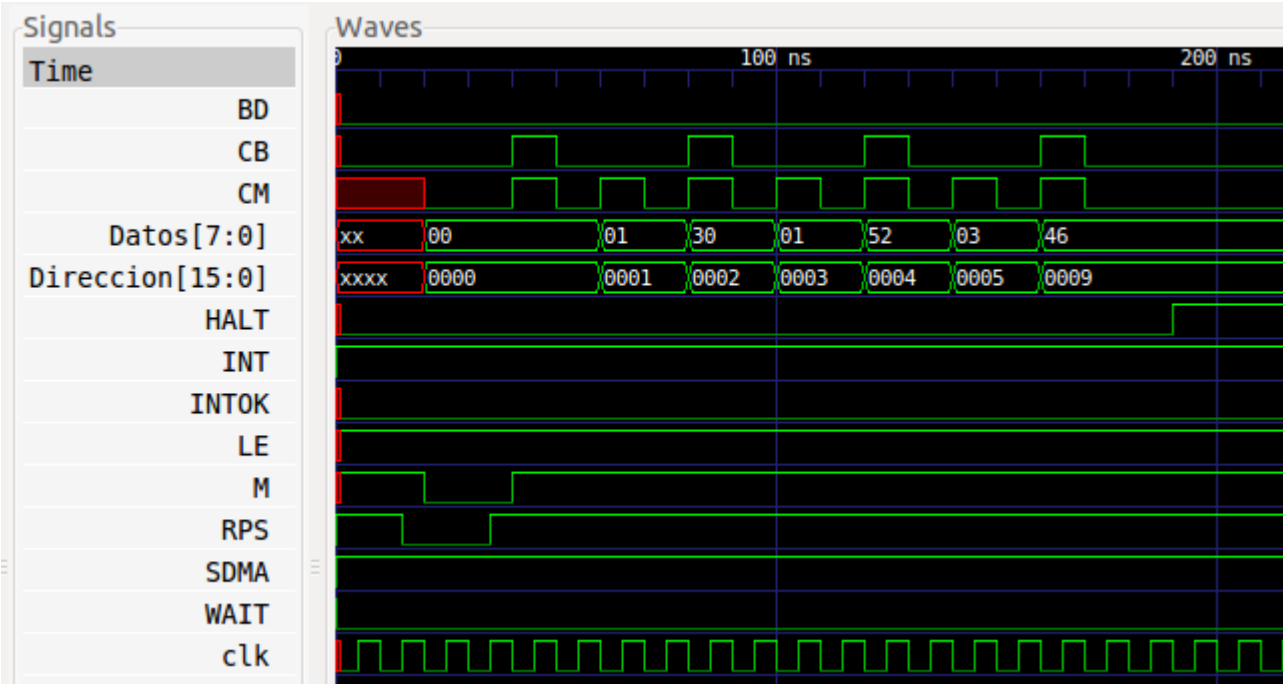

Figura 45. BPL cuando salta, correcto

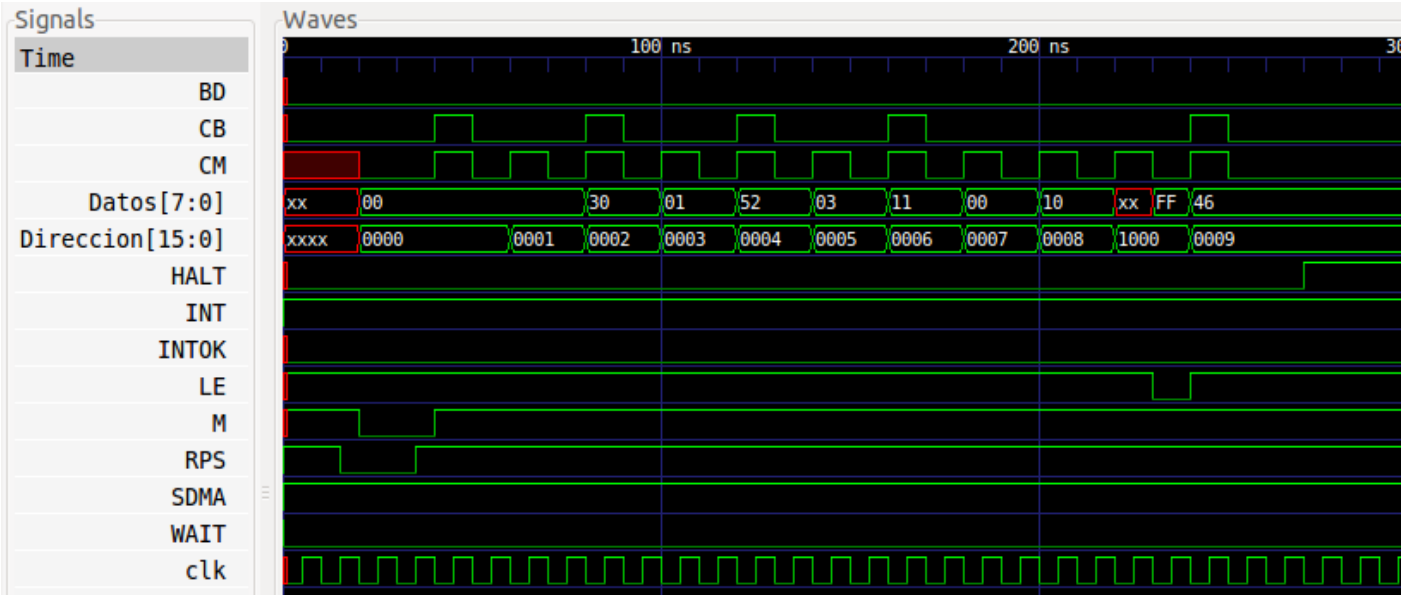

Figura 46. BPL cuando no salta, correcto

BVS (Branch if V Set)

Programa en ensamblador:

Con el fin de saltar:

```
$0000 LDA #$80
$0002 ADD #$81
$0004 BVS $0004
$0006 STA $1000
$0009 HLT
```

Con el fin de no saltar:

```
$0000 LDA #$80
$0002 ADD #$08
$0004 BVS $0004
$0006 STA $1000
$0009 HLT
```

Diagrama de temporización:

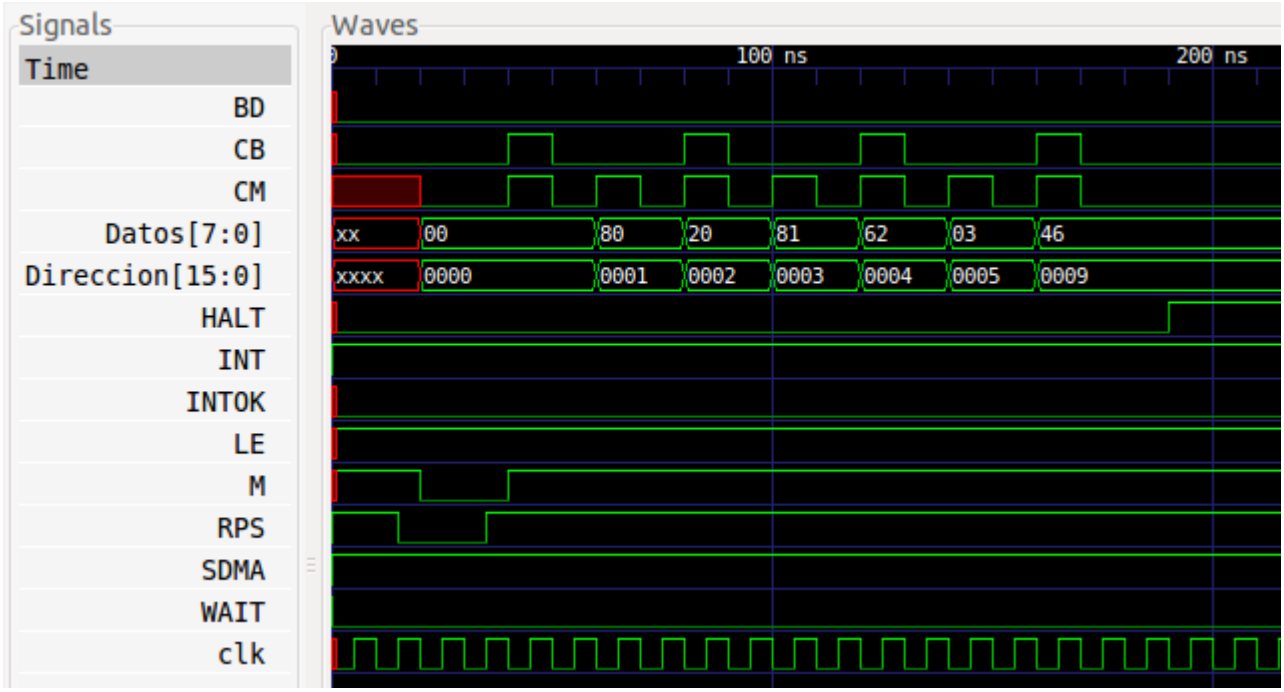

Figura 47. BVS cuando salta, correcto

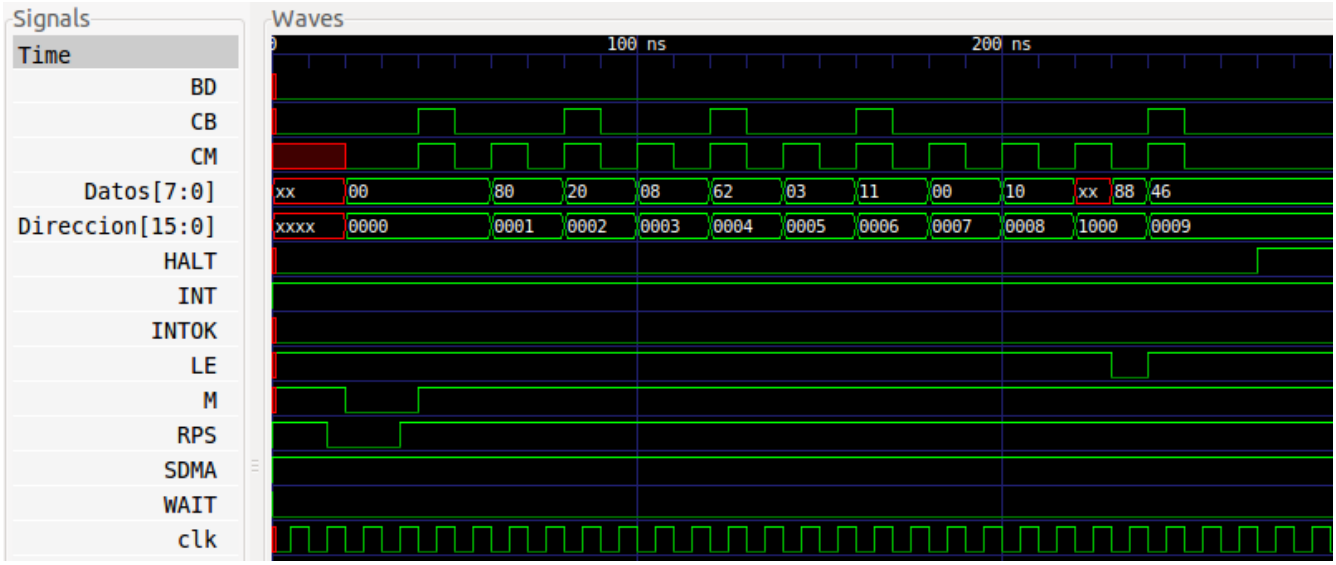

Figura 48. BVS cuando no salta, correcto

BVC (Branch if V Clear)

Programa en ensamblador:

Con el fin de saltar:

```
$0000 LDA #$80
$0002 ADD #$08
$0004 BVC $0004
$0006 STA $1000
$0009 HLT
```

Con el fin de no saltar:

```
$0000 LDA #$80
$0002 ADD #$81
$0004 BVC $0004
$0006 STA $1000
$0009 HLT
```

Diagrama de temporización:

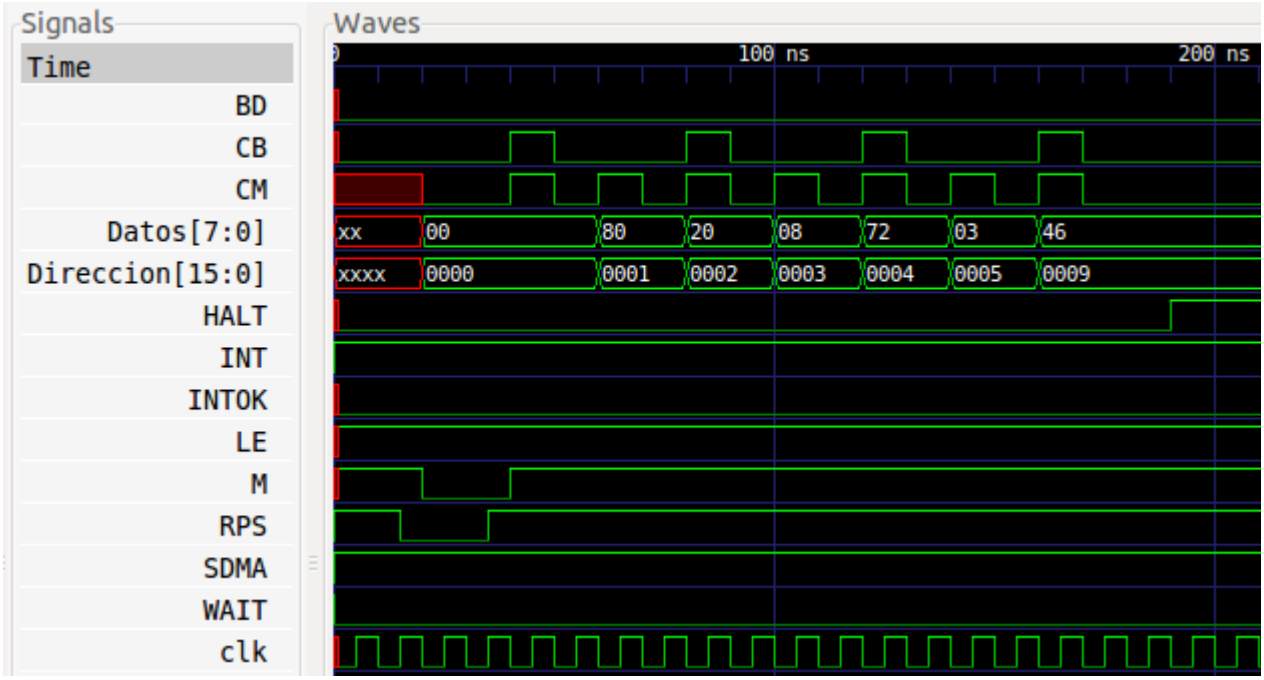

Figura 49. BVC cuando salta, correcto

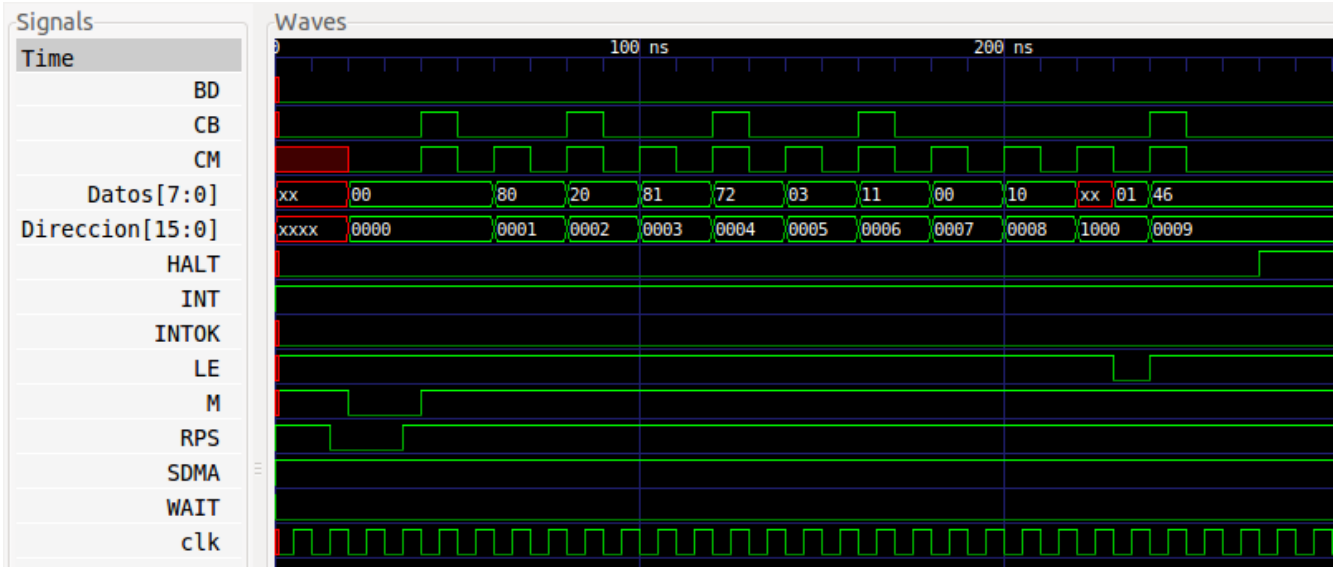

Figura 50. BVC cuando no salta, correcto

BMI (Branch if MInus)

Programa en ensamblador:

```
$0000 LDA #$00
$0002 BMI $04
$0004 DCA
$0005 JMP $0002
$0008 HLT
```

Diagrama de temporización:

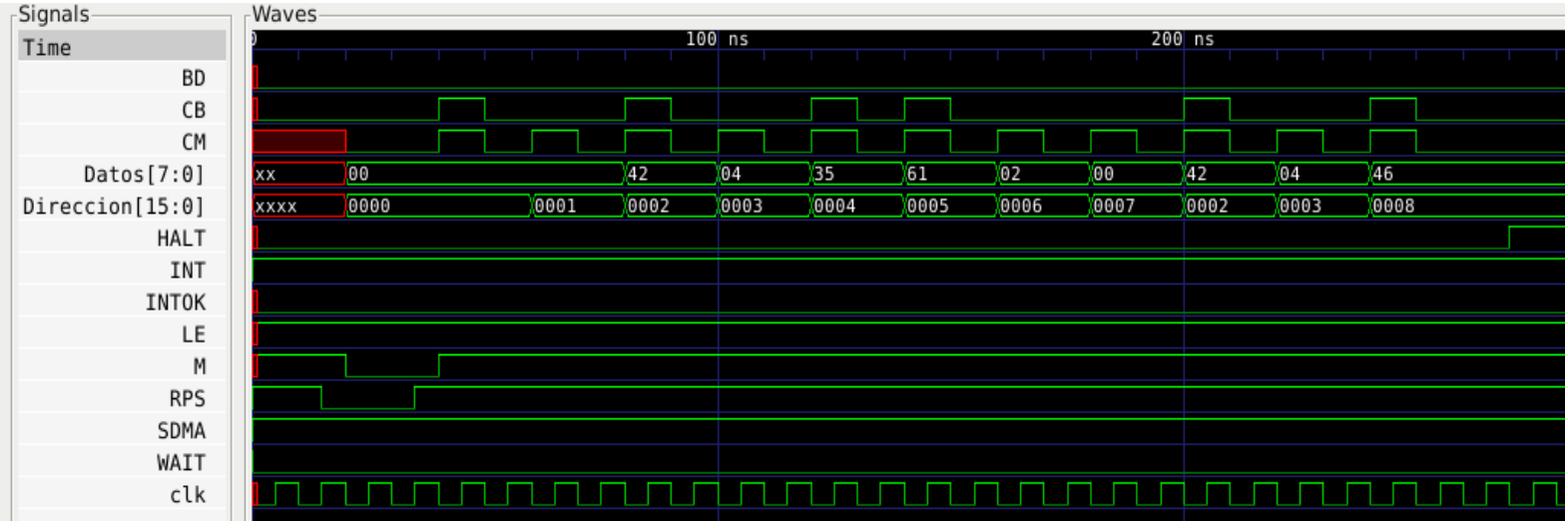

Figura 51. BMI correcto

NOP (No OPeration)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 NOP
$0003 STA $0110
$0006 HLT
```

Diagrama de temporización:

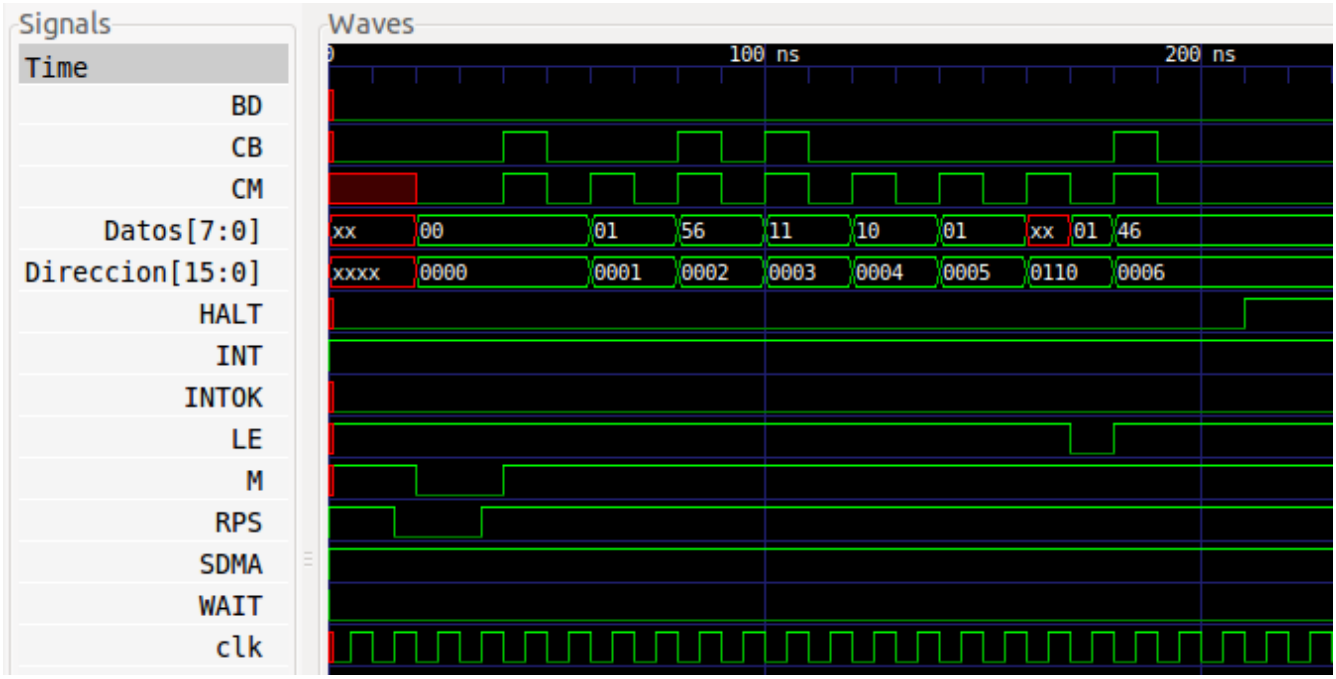

Figura 52. NOP correcto

DCA (DeCrement Accumulator)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 DCA
$0003 STA #0110
$0006 HLT
```

Diagrama de temporización:

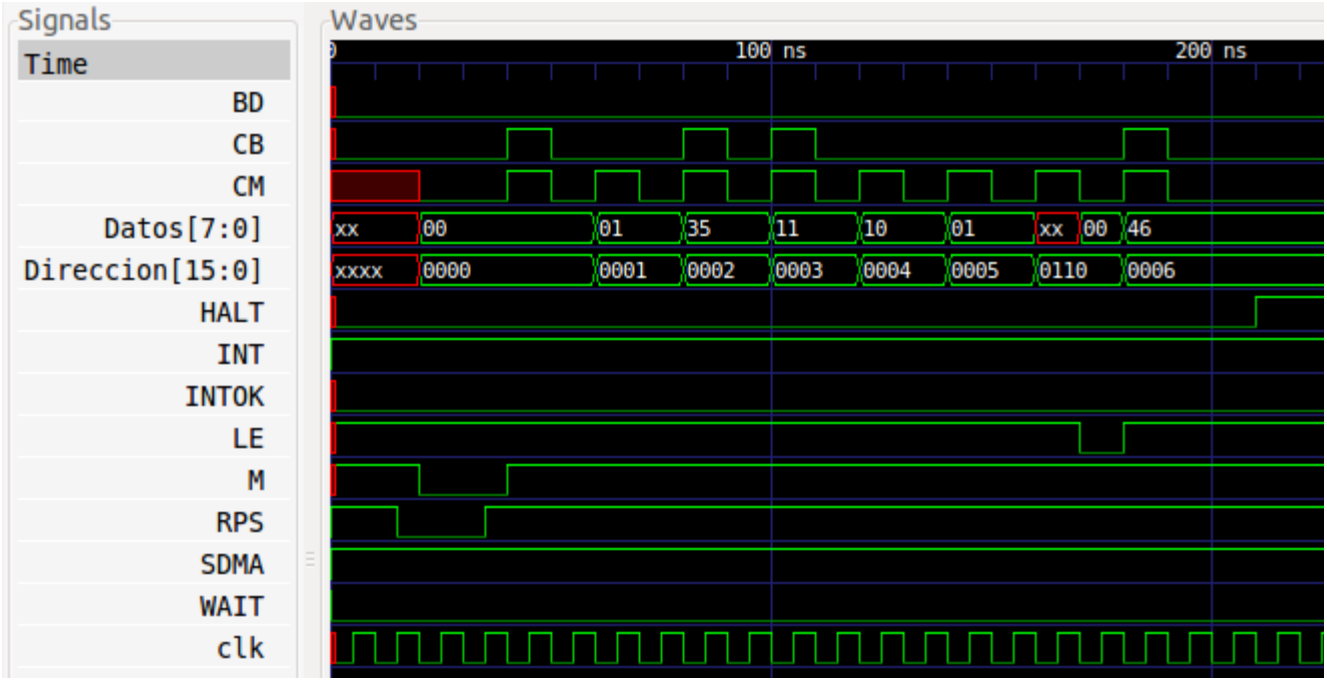

Figura 53. DCA correcto

ROL (ROtate Left)

Programa en ensamblador:

```
$0000 CLA
$0001 LDA #$05
$0003 ROL
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

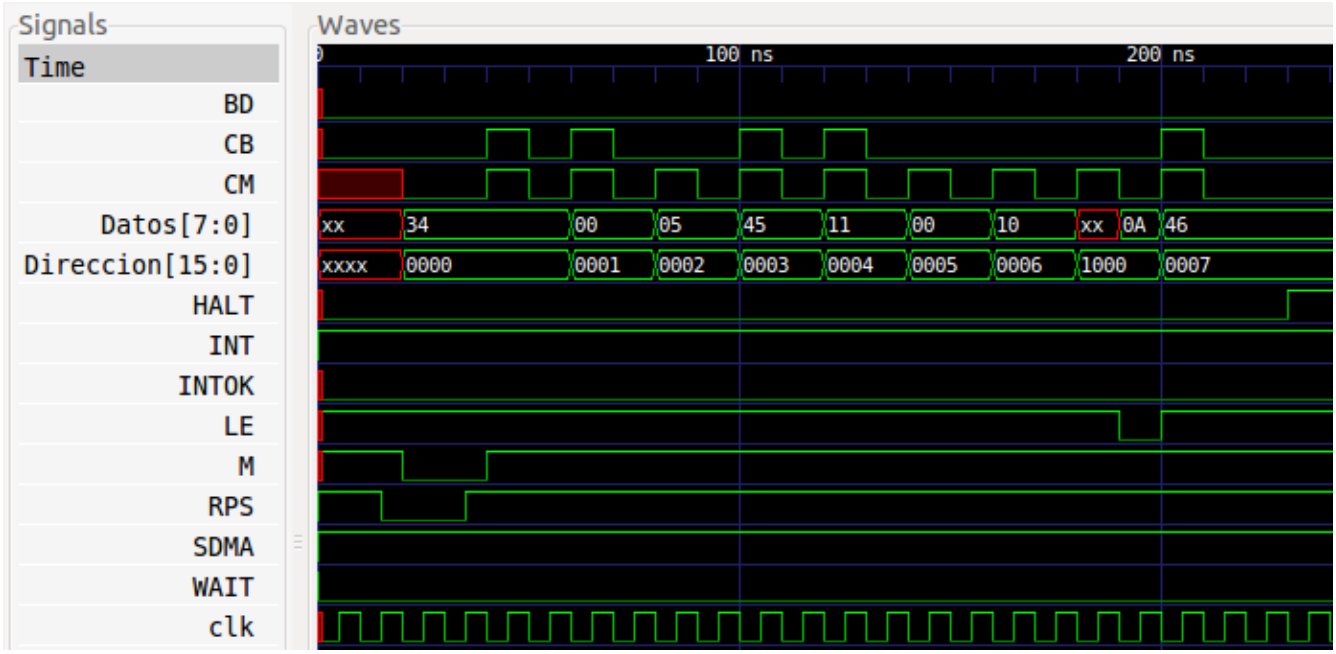

Figura 54. ROL correcto

ROR (ROtate Right)

Programa en ensamblador:

```
$0000 CLA
$0001 LDA #$06
$0003 ROL
$0004 STA $1000
$0007 HLT
```

Diagrama de temporización:

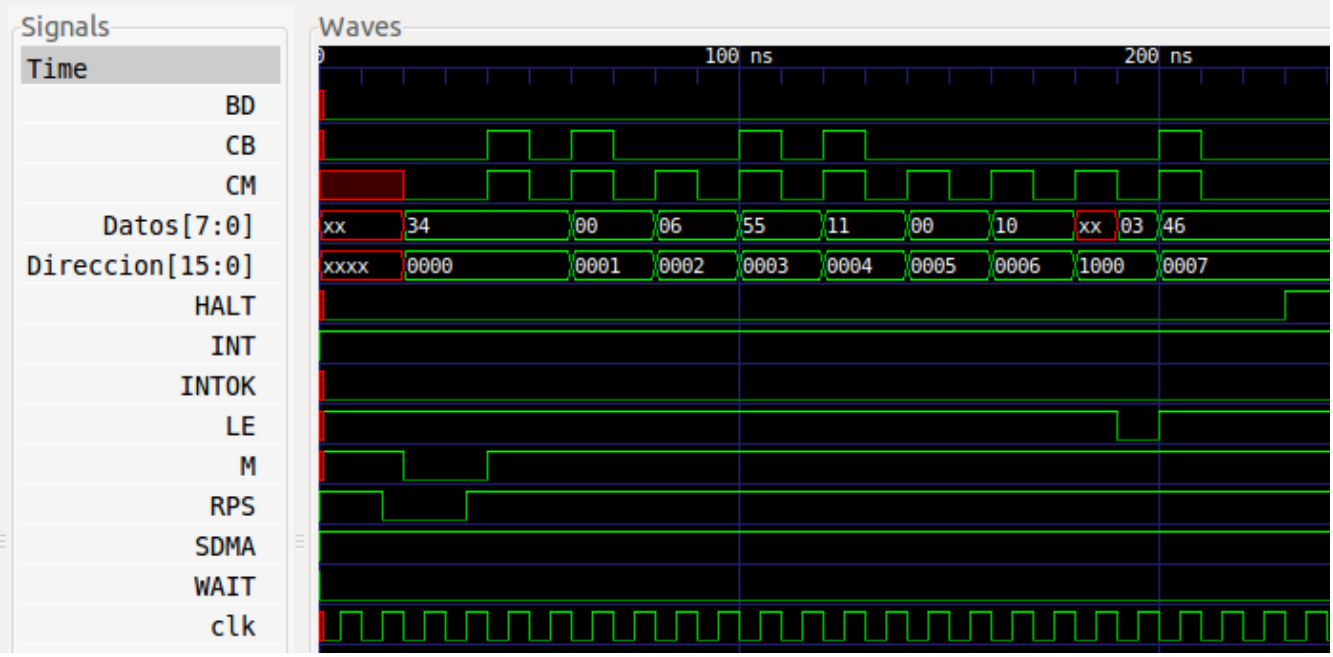

Figura 55. ROR correcto

SEC (SEt Carry flag)

Programa en ensamblador:

```
$0000 LDA #$00
$0002 TAP
$0003 SEC
$0004 PHS
$0005 HLT
```

Diagrama de temporización:

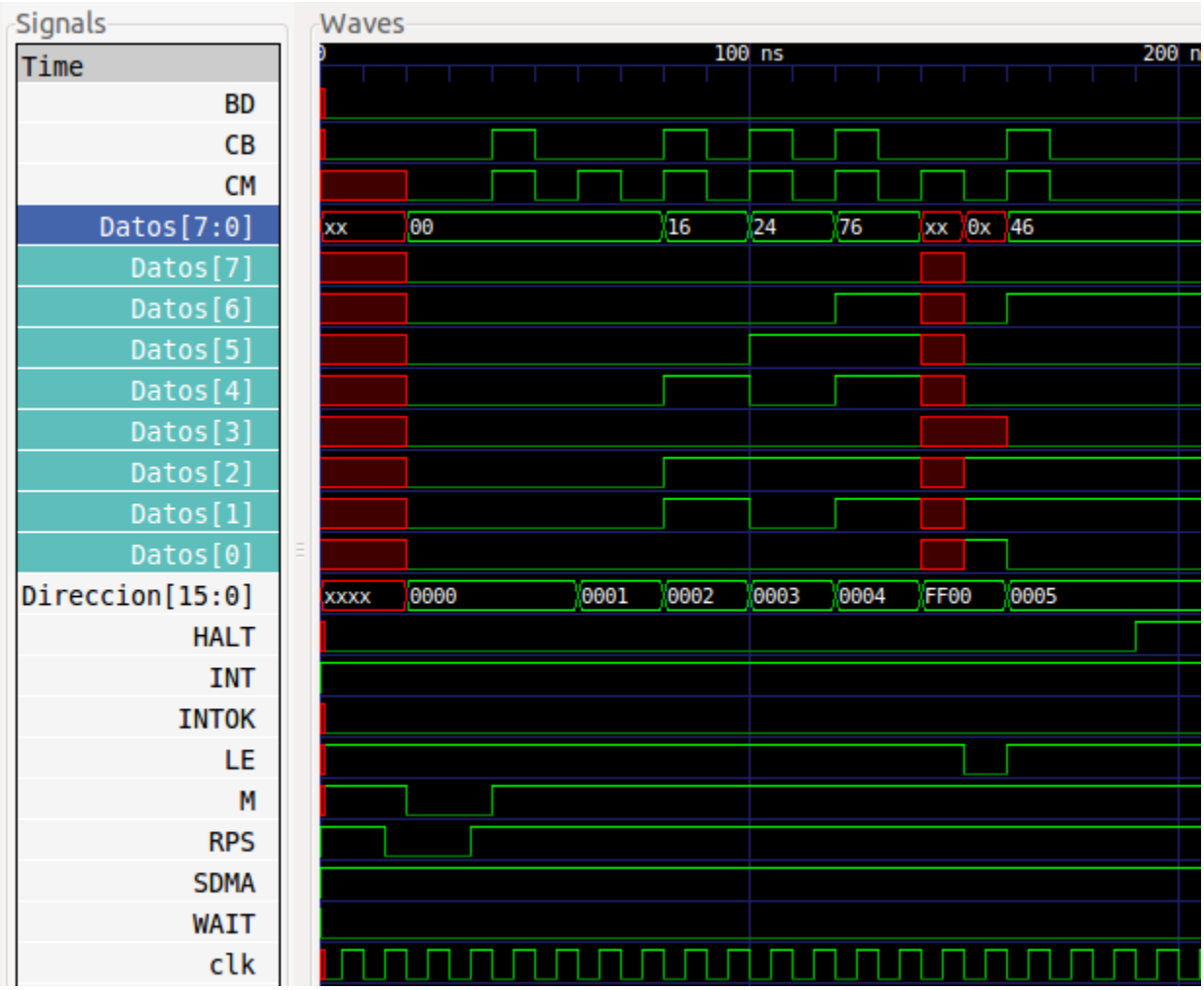

Figura 56. SEC correcto

SEI (SEt Interrupt flag)

Programa en ensamblador:

```
$0000 LDA #$00
$0002 TAP
$0003 SEI
$0004 PHS
$0005 HLT
```

Diagrama de temporización:

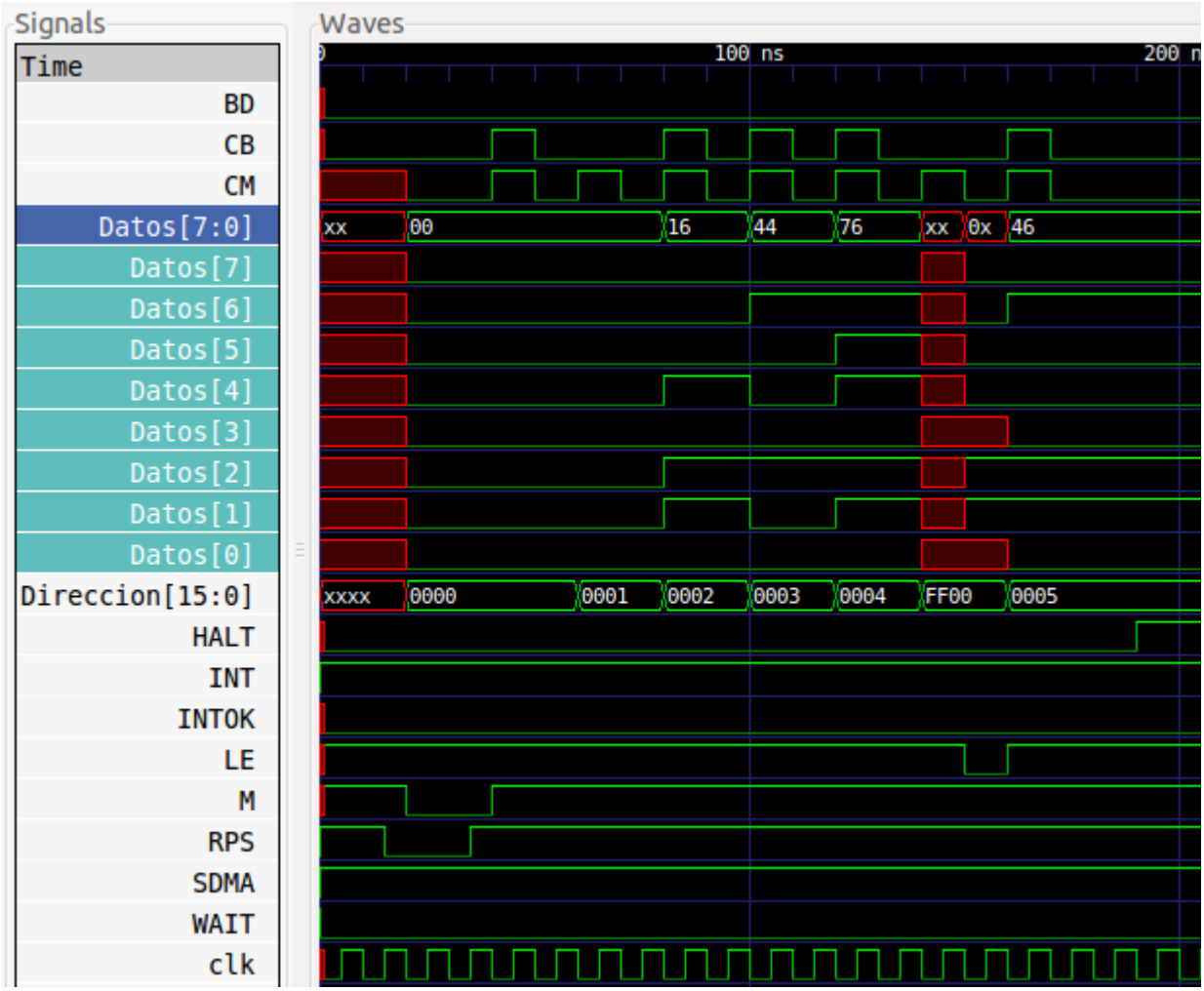

Figura 57. SEI correcto

CLI (CLear I flag)

Programa en ensamblador:

```
$0000 LDA #$00
$0002 TAP
$0003 CLI
$0004 PHS
$0005 HLT
```

Diagrama de temporización:

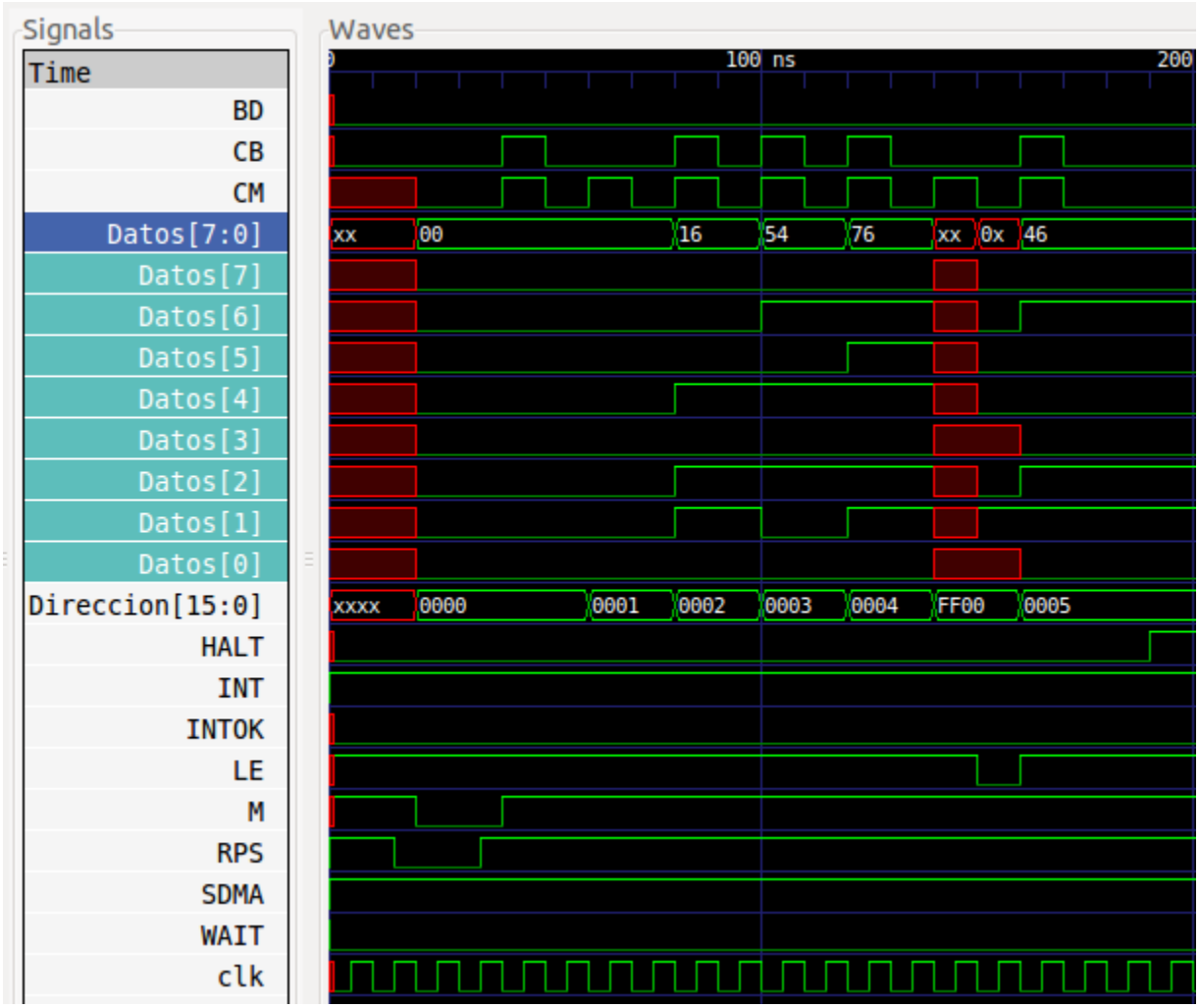

Figura 58. CLI correcto

CPA (ComPlement Accumulator)

Programa en ensamblador:

```
$0000 CLA
$0001 CPA
$0002 STA $2000
$0005 HLT
```

Diagrama de temporización:

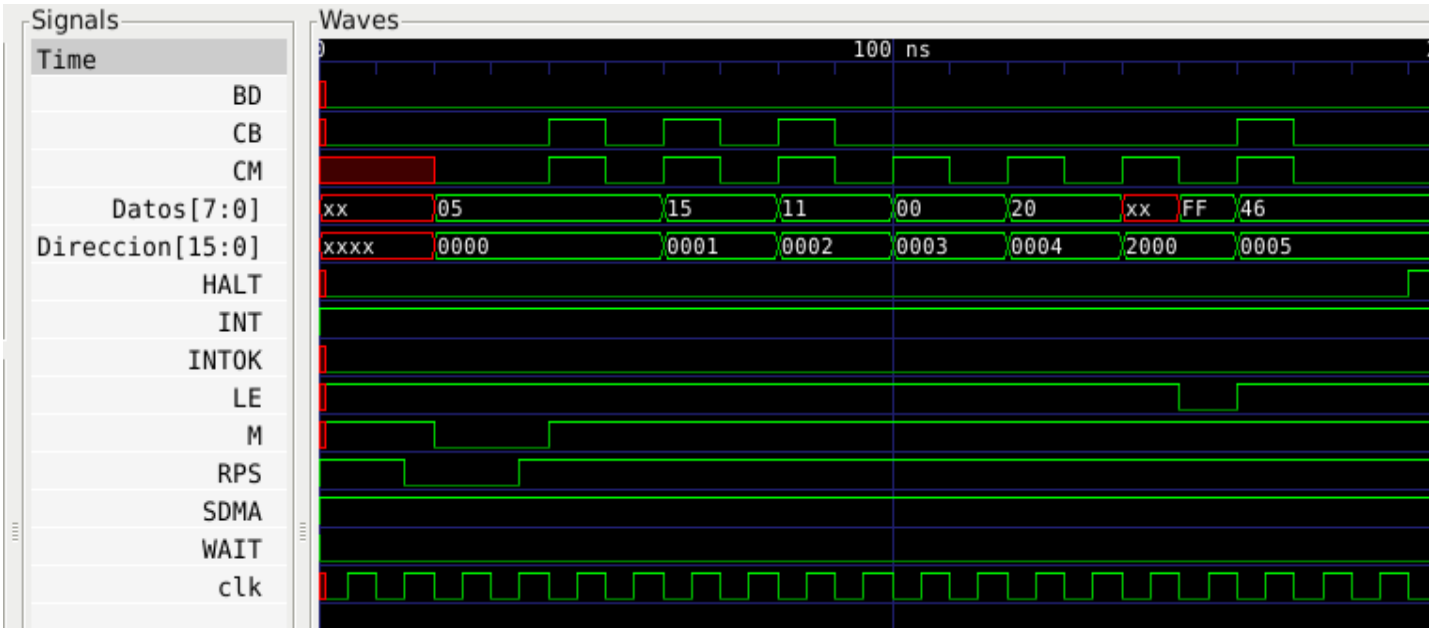

Figura 59. CPA correcto

PLA (PuLl Accumulator)

Programa en ensamblador:

```
$0000 CLA
$0001 TAP
$0002 PLA
$0003 HLT
$FFFF $09
```

Diagrama de temporización:

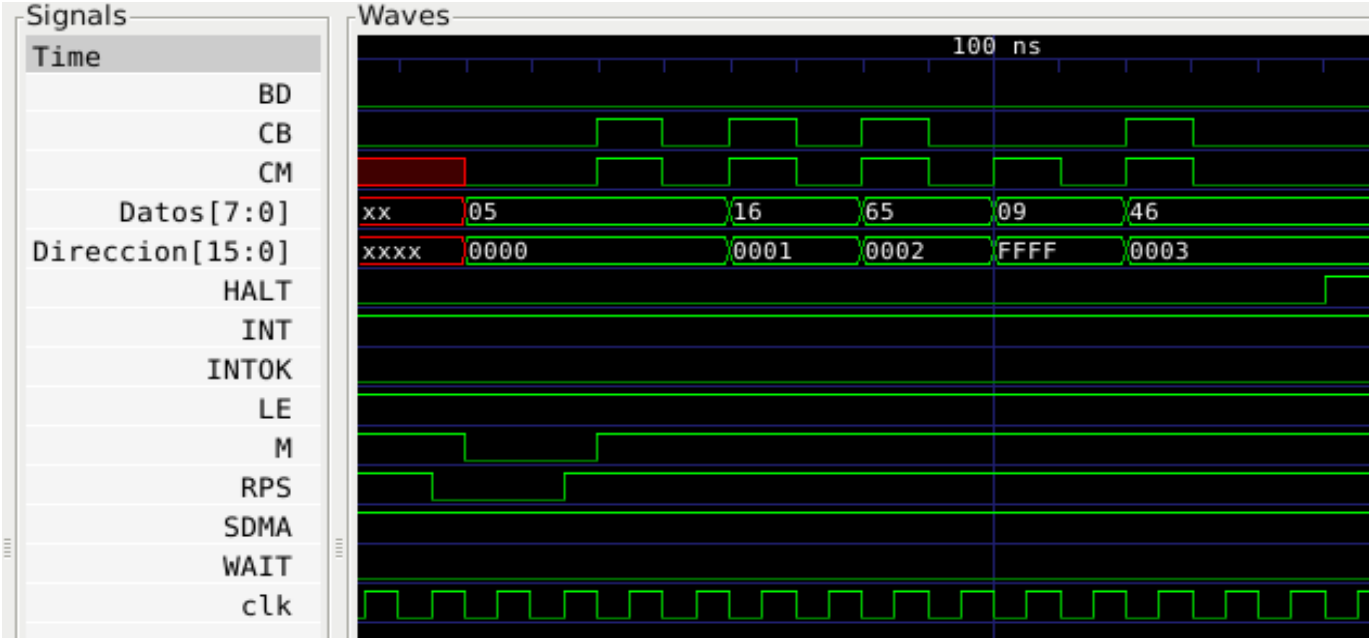

Figura 60. PLA correcto

TPA (Transfer P to A)

Programa en ensamblador:

```
$0000 LDA #$20
$0002 TAP
$0003 CLA
$0004 TPA
$0005 STA $2000
$0008 HLT
```

Diagrama de temporización:

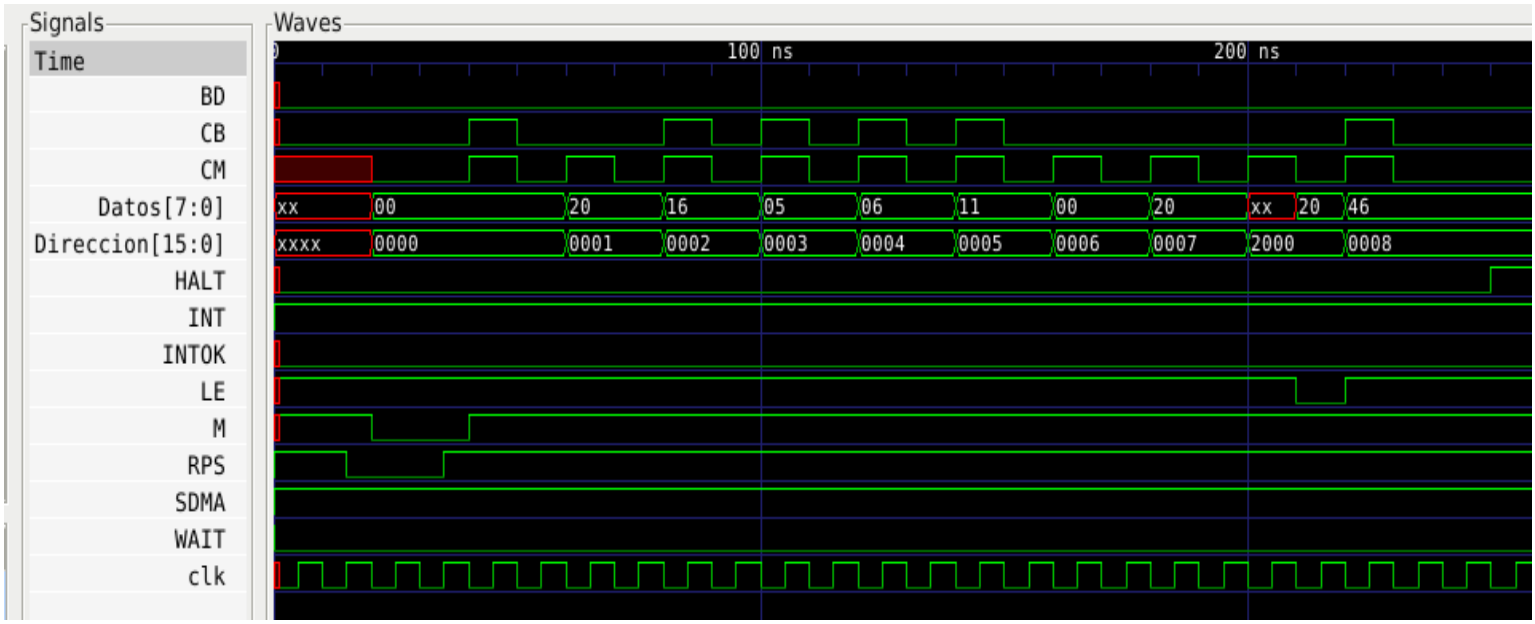

Figura 61. TPA correcto

INP (INPut)

Programa en ensamblador:

```
$0000 INP $00
$0002 HLT
$FF00 $09
```

Diagrama de temporización:

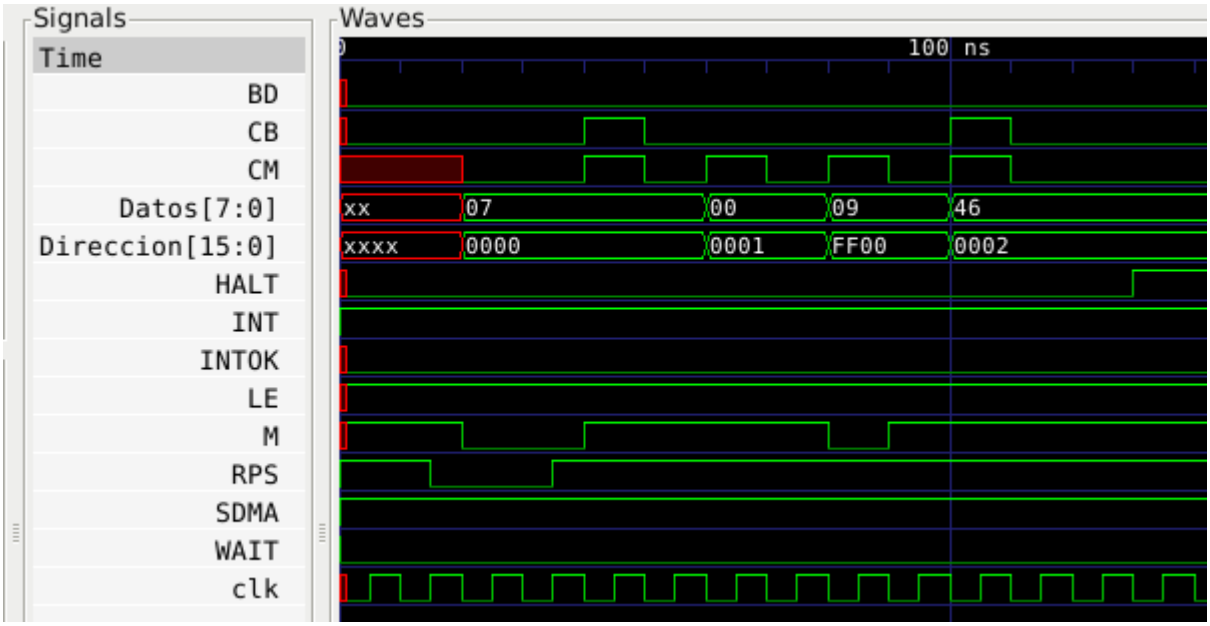

Figura 62. INP correcto

OUT (OUTput)

Programa en ensamblador:

```
$0000 LDA #$09
$0002 OUT $00
$0004 HLT
```

Diagrama de temporización:

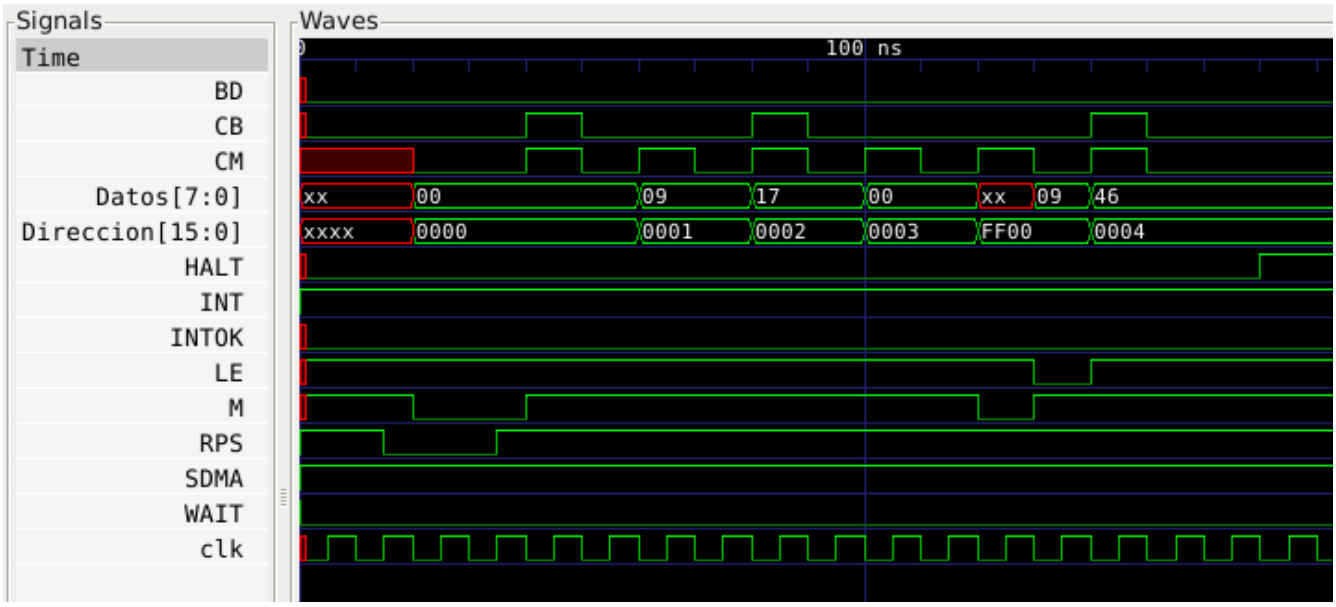

Figura 63. OUT correcto

LDAind (LoaD Accumulator indirecto)

Programa en ensamblador:

```
$0000 LDA ($1000)
$0003 HLT

$1000 $00
$1001 $20

$2000 $09
```

Diagrama de temporización:

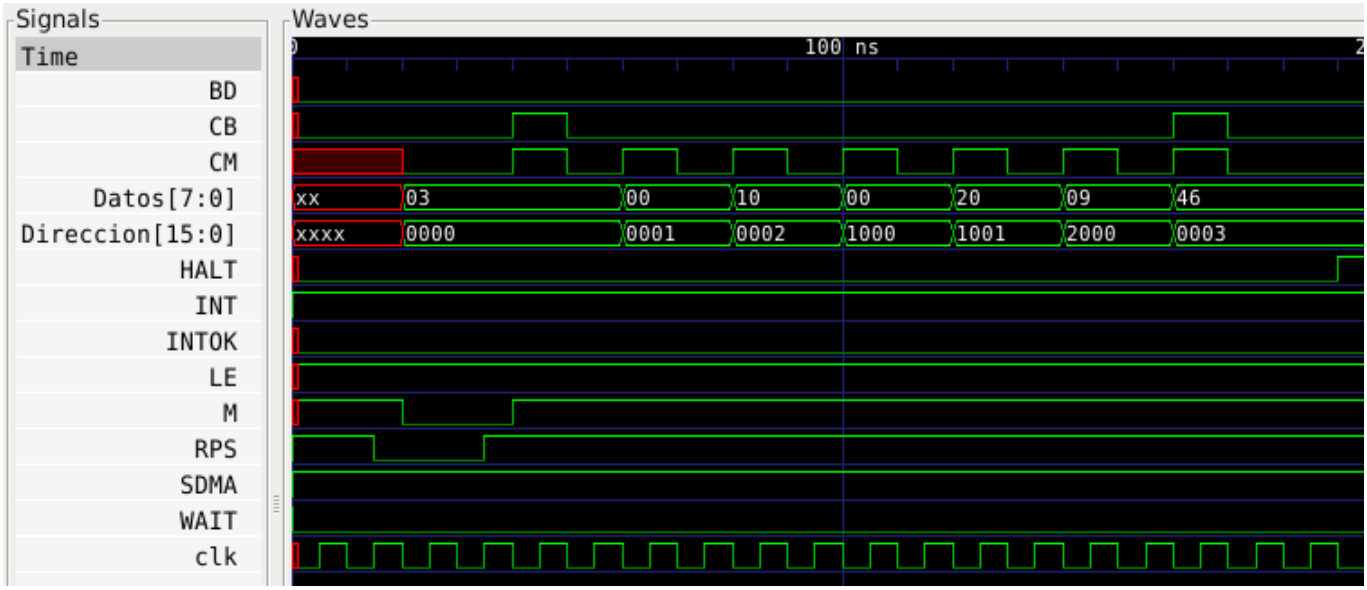

Figura 64. LDAind correcto

STAind (STore Accumulator indirecto)

Programa en ensamblador:

```
$0000 CLA
$0001 STA ($1000)
$0004 HLT

$1000 $00
$1001 $20
```

Diagrama de temporización:

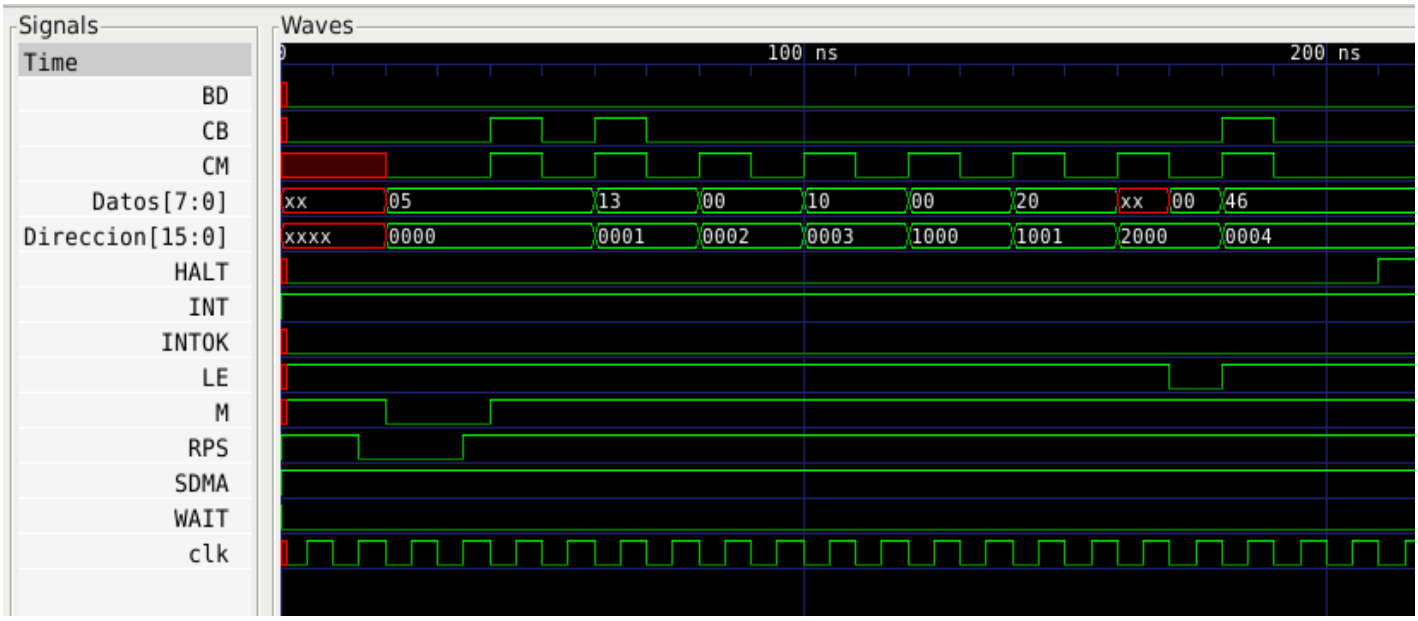

Figura 65. STAind correcto

ADDind (ADD indirecto)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 ADD ($1000)
$0005 STA $3000
$0008 HLT

$1000 $00
$1001 $20

$2000 $09
```

Diagrama de temporización:

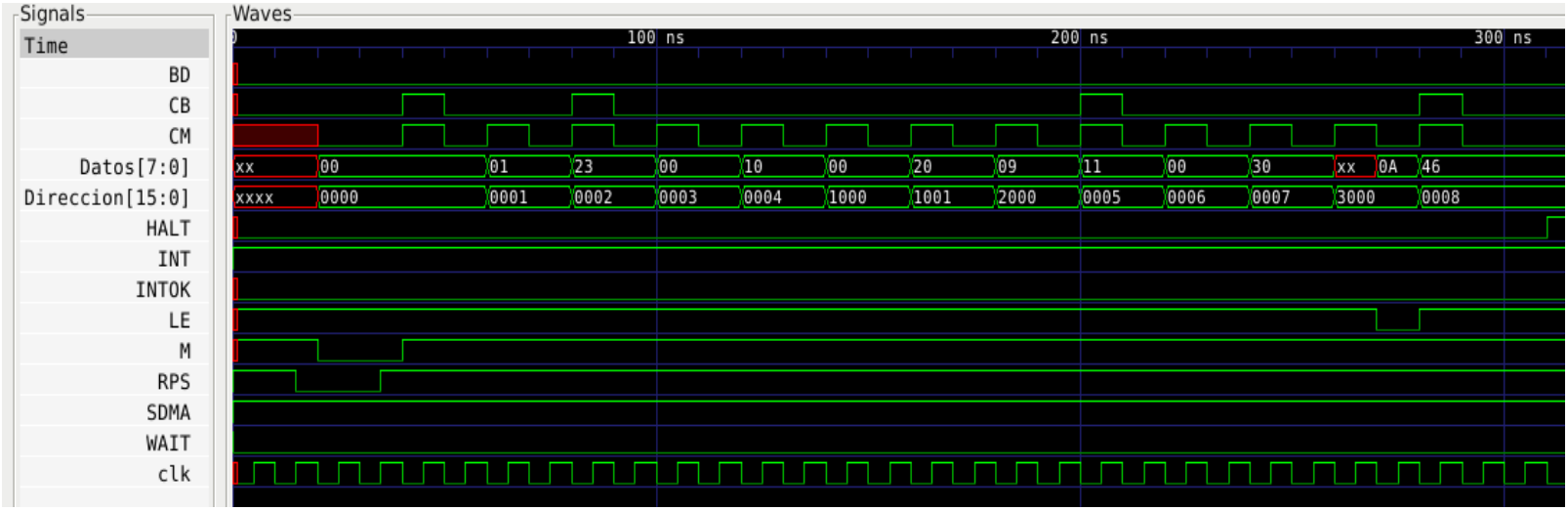

Figura 66. ADDind correcto

SUBind (SUBtract indirecto)

Programa en ensamblador:

```
$0000 LDA #$01
$0002 SUB ($1000)
$0005 STA $3000
$0008 HLT

$1000 $00
$1001 $20

$2000 $09
```

Diagrama de temporización:

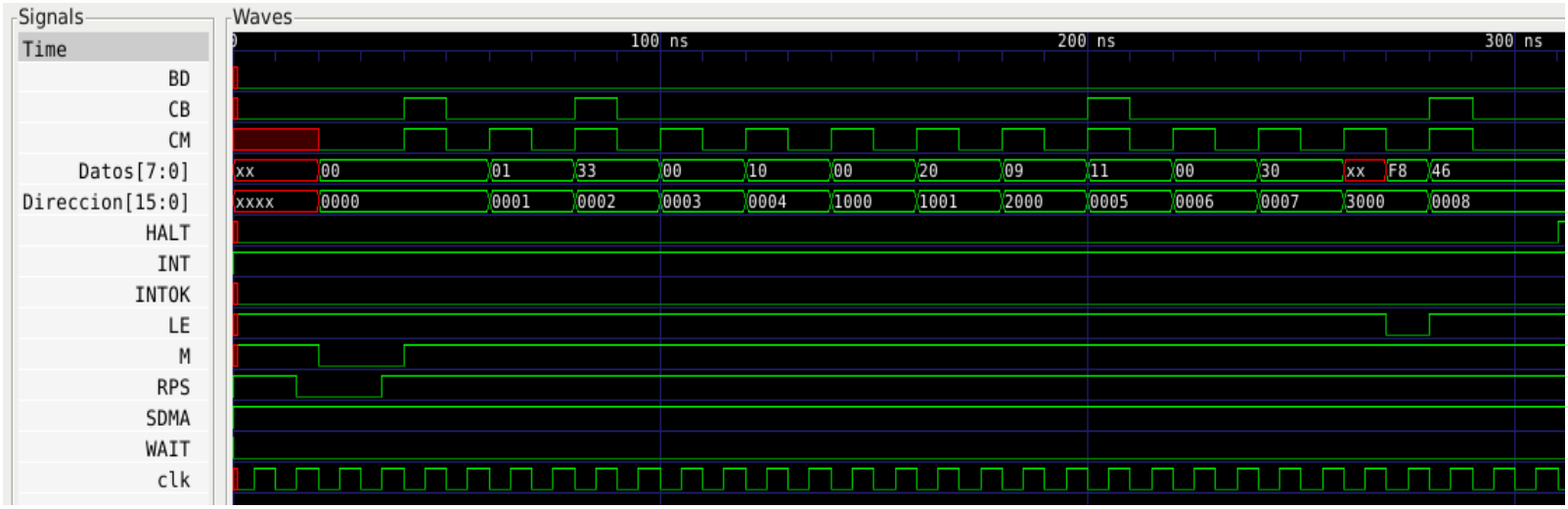

Figura 67. SUBind correcto