

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторної роботи з дисципліни
“Технології проектування комп'ютерних систем”
для студентів спеціальностей 8.05010201 "Комп'ютерні системи і
мережі", 8.05010203 "Спеціалізовані комп'ютерні системи"
всіх форм навчання

Частина 4

2015

Методичні вказівки до виконання лабораторної роботи з дисципліни “Технології проектування комп’ютерних систем” для студентів спеціальностей 8.05010201 “Комп’ютерні системи і мережі”, 8.05010203 “Спеціалізовані комп’ютерні системи” всіх форм навчання. Частина 4. /Укл. А.В. Сирота. - Запоріжжя: ЗНТУ, 2015. – 54 с.

Рекомендовано до видання НМК спеціальності як методичні вказівки до лабораторних робіт з дисципліни „Технології проектування комп’ютерних систем” усіх форм навчання.

Укладач: А.В. Сирота, ст. викладач.

Рецензент: Р.К. Кудерметов, доцент, к.т.н.

Відповідальний за випуск: А.В. Сирота, ст. викладач.

Затверджено
на засіданні кафедри КСМ

Протокол № 6
від 02.02.2015

ЗМІСТ

1 ПРОЕКТУВАННЯ МОДЕЛІ МІКРОПРОЦЕСОРА	4
1.1 Опис об'єкту проектування	4
1.2 Внутрішня архітектура універсального мікропроцесора.....	4
2 ПРОЕКТУВАННЯ МОДЕЛІ МІКРОПРОЦЕСОРА	13
2.1 Система команд мікропроцесора.....	13
2.2 Інформаційний тракт мікропроцесора	14
2.2.1 Команда NOP	14
2.2.2 Команда ADD	15
2.2.3 Команда PASS.....	16
2.2.4 Команда INC	16
2.2.5 Команда ZERO	17
2.2.6 Команда STORE.....	18
2.2.7 Команда LOADI.....	18
2.3 Тракт керування мікропроцесора	19
2.3.1 Скидання і початок роботи мікропроцесора	20
2.3.2 Зчитування та декодування команди.....	21
2.3.3 Інкремент лічильника команд.....	21
2.4 Моделі мікропроцесора	22
2.4.1 Модель регістра	22
2.4.2 Модель регістра з трьома станами	23
2.4.3 Модель регістрової пам'яті МП.....	25
2.4.4 Пакет глобальних визначень проекту.....	26
2.4.5 Модель АЛП	27
2.4.6 Складання моделі керуючого пристрою.....	28
2.4.7 Модель МП	37
2.4.8 Файл верхнього рівня VHDL-проекту.....	41
2.5 Введення моделей в систему Active-HDL	43
2.6 Функціональне моделювання МП.....	43
2.7 Зміст звіту	45
2.8 Контрольні питання	46
ЛІТЕРАТУРА	47
Додаток А	48

1 ПРОЕКТУВАННЯ МОДЕЛІ МІКРОПРОЦЕСОРА

Лабораторна робота № 6

1.1 Опис об'єкту проектування

Інтерфейс мікропроцесора.

Мікропроцесор (МП) має необхідну кількість сигналів (портів), що створюють *мікропроцесорний інтерфейс* [8]:

clock	вх.	(1 біт)	тактовий сигнал;
reset	вх.	(1 біт)	сигнал скидання;
address	вих.	(16 біт)	зовнішня магістраль адреси;
data	вх./вих.	(16 біт)	загальна магістраль даних;
rw	вих.	(1 біт)	завдання типу операції з пам'яттю: читання / запис;
vma	вих.	(1 біт)	сигнал дозволу операцій з пам'яттю
ready	вх.	(1 біт)	підтвердження пам'яттю даних на магістралі data (0:15).

1.2 Внутрішня архітектура універсального мікропроцесора

Структура спрощеного 16-розрядного МП приведена на рис. 1.1. Короткий опис його архітектурних компонентів в табл. 1.1.

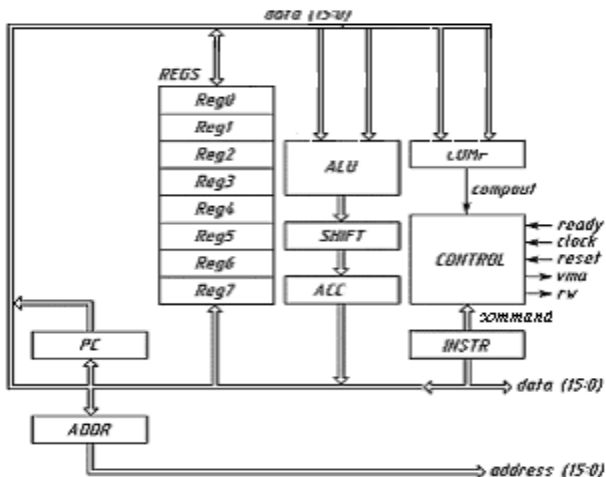


Рисунок 1.1- Внутрішня архітектура МП

Таблиця 1.1 - Архітектурні компоненти МП

Блок	Назва	Призначення
ACC	Акумулятор	Зберігає результат роботи МП після обробки поточної команди
ADDR	Регістр адреси	Зберігає адресу комірки під час звернення до пам'яті
ALU	Арифметично-логічний пристрій	Виконує ряд логічних і арифметичних операцій
COMP	Компаратор	Перевіряє умови виконання команд умовного переходу
CONTROL	Пристрій керування	Виконує загальне керування роботою усіх блоків МП
INSTR	Регістр команди	Зберігає чергову команду на етапі декодування
PC	Програмний лічильник	Містить адресу команди, що підлягає виконанню
REGS	Регістрова пам'ять	Вісім 16-розрядних регістрів, призначених для зберігання операндів і проміжних результатів
SHIFT	Регістр зсуву	Виконує операції зсуву результату роботи АЛП

Обмін даними між блоками МП забезпечується *магістралями*:

- data (15:0)** – загальна магістраль даних, яка використовується також для завантаження операндів до арифметично-логічного пристрою (АЛП) і компаратора;
- command(15:0)** – магістраль чергової команди регістра машинної команди (РМК);
- aluout (15:0)** – магістраль результату АЛП;
- shiftout (15:0)** – магістраль результату регістра зсуву.

Ці магістралі забезпечують збалансоване виконання основних операцій читання / запису даних МП (див. табл. 1.2).

Таблиця 1.2 - Використання магістралей МП

	Операція	Магістраль
T1	Завантаження чергової команди до РМК	data (15:0)
T2	Завантаження машинної команди до пристрою керування,	command (15:0)
T3	Завантаження операнда 1 до АЛП / компаратора	data (15:0)
T4	Завантаження операнда 2 до АЛП / компаратора	data (15:0)
T5	Запис результату АЛП / регістра зсуву до акумулятора	aluout (15:0), shiftout (15:0)
T6	Запис даних до регістрової пам'яті	data (15:0)
T7	Завантаження вмісту ЛК до АЛП	data (15:0)
T8	Завантаження нового значення ЛК	data (15:0)
T9	Завантаження нового значення регістра адреси	data (15:0)

1.3 Порядок розробки моделі мікропроцесора

Алгоритм розробки моделі мікропроцесора.

- 1 Скласти програму мовою асемблера, виконати кодування машинних команд.
- 2 Вибрати відповідно до заданої програми систему команд і внутрішню архітектуру МП.
- 3 Створити модель регістрів: акумулятора (компонент **ACC**), адреси (компонент **ADDR**), ЛК (компонент **PC**), РМК (компонент **INSTR**). У даному випадку можлива реалізація цих компонентів за допомогою **уніфікованих** об'єктів: моделі звичайного регістра (файл **reg.vhd**) і регістра, який має третій стан (файл **trireg.vhd**). Отримати часові діаграми функціонального моделювання цих регістрів.
- 4 Створити модель файлу регістрів **REGS** – файл **regarray.vhd**.
Отримати часові діаграми функціонального моделювання цього блоку.
- 5 Створити модель арифметично-логічного пристрою **ALU** – файл **alu.vhd**.
Отримати часові діаграми функціонального моделювання цього блоку.
- 6 Скласти **прямую структурну таблицю (ПСТ)** автомата.
- 7 Скласти модель пристрою керування **CONTROL** – файл **control.vhd**.
Отримати часові діаграми функціонального моделювання цього блоку.
- 8 У вікні VHDL редактора (**HDL Editor**) ввести опис **cpu** – моделі мікропроцесора, сутність якого складається з декларацій зовнішніх сигналів, а виконавча частина - з декларацій компонентів системи (**CONTROL, ALU, ACC, REGS, INSTR, ADDR, PC**) з їх сигналами та з'єднань компонентів, виконаних за допомогою операторів **PORT MAP**.
- 9 Отримати часові діаграми функціонального моделювання цього блоку.
- 10 Взяти із лаб.5 опис мікрокомп'ютера **top.vhd**. Замість імітатора **icpu** вставити модель мікропроцесора **cpu**.
- 11 Отримати часові діаграми функціонального моделювання роботи мікрокомп'ютера під час виконання програми.

Розглянемо програму складання двох чисел, які знаходяться за адресами **0001h** та **0002h** і результат заноситься за адресою **0010h**. Для виконання цієї програми розробимо відповідну систему команд і внутрішню архітектуру МП.

Лістинг 1– Програма складання двох чисел

```
0000    LOADI    R1,    #0001h ; запис першого операнда до  
регістра Reg1
```

```

0002  LOADI  R2,  #0002h ; запис другого операнда до
регiстра Reg2
0002  LOADI  R4,  #0010h ; запис адреси результату до
регiстра Reg4
0004  AAD     R3,  R2, R1 ; запис [R2] + [R1] в ре-
гiстр Reg3
0006  STORE  @R3, R4      ; копіювання вмісту R3 ([R3])
до місця збере-

```

ження в ОП([Reg4])

Складемо бінарний код інструкцій програми:

```

0000  "10110 00 001 000 000" ; LOADI R1,
0001  "0000 0000 0000 0001" ; #0001h
0002  "10110 00 010 000 000" ; LOADI R2,
0003  "0000 0000 0000 0010" ; #0002h
0004  "10110 00 010 000 000" ; LOADI R4,
0005  "0000 0000 0001 0000" ; #0002h
0006  "00101 00 011 010 001" ; ADD, R3, R2, R1
0007  "11000 00 001 000 011" ; STORE @R4, R3

```

Відповідно до програми спростимо систему команд і внутрішню архітектуру МП.

Система команд МП, яка буде виконуватись в цьому МП, наведена в табл. 1.3.

Таблиця 1.3 - Система команд МП

OPCODE	Асемблер	Операція	Тип	Приклад
00000	NOP	Немає операції	1	NOP
00101	ADD	Додавання	1	ADD R1, R2, R3
00110	INC	Інкремент	1	INC R1, R2
01001	ZERO	Нуль	1	ZERO R1
10001	PASS	Передача без змін	1	PASS R1, R2
10111	LOADI	Завантаження регістра значенням	1	LOADI R3, #10
11000	STORE	Збереження значення в пам'яті	1	STORE @R3, R1

Внутрішня архітектура МП, що розглядається, наведена на рис. 2.

Особливості внутрішньої архітектури цього МП:

- відсутні компаратор та регістр зсуву;
- для завантаження першого і другого операндів до АЛП використовується магістраль **data (15:0)**;
- введена внутрішня магістраль даних **aluout** між АЛП і акумулятором.

Конфігурація мікропроцесорного інтерфейсу (див. рис.1.2), логічний протокол взаємодії із зовнішньою пам'яттю **лишилися** без змін.

Незмінним залишається і порядок розробки моделі мовою VHDL засобами САПР Active-HDL.

Обмін даними між блоками МП забезпечується *магістралями*:

data (15:0) – загальна магістраль даних, що безпосередньо виведена до порту **data (15:0)**;

aluout (15:0) – магістраль результату АЛП;

command(15:0) – магістраль для завантаження бінарного коду машинної команди до пристрою керування.

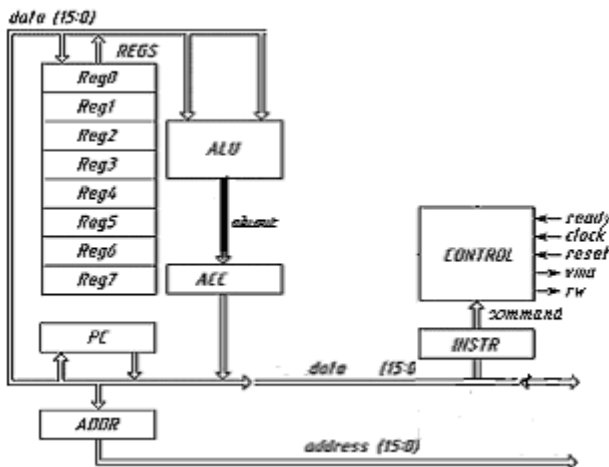


Рисунок 1.2 - Варіант внутрішньої архітектури МП

Ці магістралі забезпечують збалансоване виконання основних операцій читання / запису даних МП (див. табл. 1.4).

Таблиця 1.4 - Використання магістралей МП

	Операція	Магістраль
T1	Завантаження чергової команди до РМК (INSTR)	<i>data (15:0)</i>
T2	Завантаження машинної команди до керуючого пристрою	<i>command(15:0)</i>
T3	Завантаження операнда 1 до АЛП (ALU)	<i>data (15:0)</i>
T4	Завантаження операнда 2 до АЛП (ALU)	<i>data (15:0)</i>
T5	Запис результату АЛП до акумулятора	<i>aluout (15:0),)</i>
T6	Запис даних до регістрової пам'яті	<i>data (15:0)</i>
T7	Завантаження змісту ЛК (PC) до АЛП (ALU)	<i>data (15:0)</i>
T8	Завантаження нового значення ЛК / регістра адреси	<i>data (15:0)</i>
T9	Завантаження нового значення регістра адреси	<i>data (15:0)</i>
T10	Виведення значення регістра адреси на шину адреси	<i>addrs (15:0)</i>
T11	Виведення даних на зовнішню шину даних	<i>data (15:0)</i>

Акумулятор ACC є 16-розрядний регістр, що призначений для збереження результату АЛП. Умовне графічне позначення акумулятора наведено на рис. 3,а. Призначення портів акумулятора:

aluout	вх.	(16 біт)	данні магістралі результату АЛП;
AcregRd	вх.	(1 біт)	сигнал читання вмісту акумулятора;
AcregWr	вх.	(1 біт)	сигнал запису даних до акумулятора;
data	вих.	(16 біт)	загальна магістраль даних.

Регістр адреси ADDR є 16-розрядний регістр, що призначений для збереження адреси даних або команди в пам'яті. Умовне графічне позначення регістра адреси наведено на рис. 3,б. Призначення портів регістра адреси:

data	вх.	(16 біт)	загальна магістраль даних;
addrWr	вх.	(1 біт)	сигнал запису даних загальної магістралі даних;
address	вих.	(16 біт)	зовнішня магістраль адреси.

Арифметично-логічний пристрій ALU це комбінаційна схема, що виконує ряд логічних і арифметичних операцій. Розрядність операндів – 16 біт.

Умовне графічне позначення АЛП наведено на рис.1.3в. Призначення портів арифметично-логічного пристроя:

data	вх.	(16 біт)	дані спочатку першого, а потім другого операндів;
we1	вх.	(1 біт)	сигнал завантаження даних першого операнда;
we2	вх.	(1 біт)	сигнал завантаження даних другого операнда;
alusel	вх.	(5 біт)	сигнал вибору функції АЛП;
aluout	вих.	(16 біт)	магістраль результату АЛП.

Пристрій керування CONTROL це керуючий С-автомат, що виконує декодування машинної команди і формує сигнали, що керують режимом роботи усіх блоків МП. Умовне графічне позначення пристроя керування наведено на рис. 3,д. Призначення портів пристроя керування:

command	вх.	(16 біт)	бінарний код машинної команди;
Ready	вх.	(1 біт)	підтвердження даних на магістралі <i>data</i> (0:15).
Clock	вх.	(1 біт)	тактовий сигнал;
Reset	вх.	(1 біт)	сигнал скидання;
vma	вих.	(1 біт)	підтвердження адреси на магістралі <i>address</i> (0:15);
rw	вих.	(1 біт)	завдання типу операції з пам'яттю: читання / запис;
ACregRd	вих.	(1 біт)	дозвіл читання вмісту акумулятора;
ACregWr	вих.	(1 біт)	сигнал запису даних до акумулятора;
PCregRd	вих.	(1 біт)	дозвіл читання вмісту ЛК;
PCregWr	вих.	(1 біт)	сигнал запису даних до ЛК;
addrWr	вих.	(1 біт)	сигнал запису даних загальної магістралі даних;
instrWr	вих.	(1 біт)	сигнал запису даних до РМК;
we1	вих.	(1 біт)	сигнал завантаження даних першого операнда;
we2	вих.	(1 біт)	сигнал завантаження даних другого операнда;
alusel	вих.	(5 біт)	сигнал вибору функції АЛП;
regsel1	вих.	(3 біти)	сигнал вибору регістра першого операнда (основний);
regsel2	вих.	(3 біти)	сигнал вибору регістра другого операнда (додатковий);
regRd	вих.	(1 біт)	дозвіл читання вмісту регістрової пам'яті;
regWr	вих.	(1 біт)	сигнал запису даних до регістрової пам'яті.

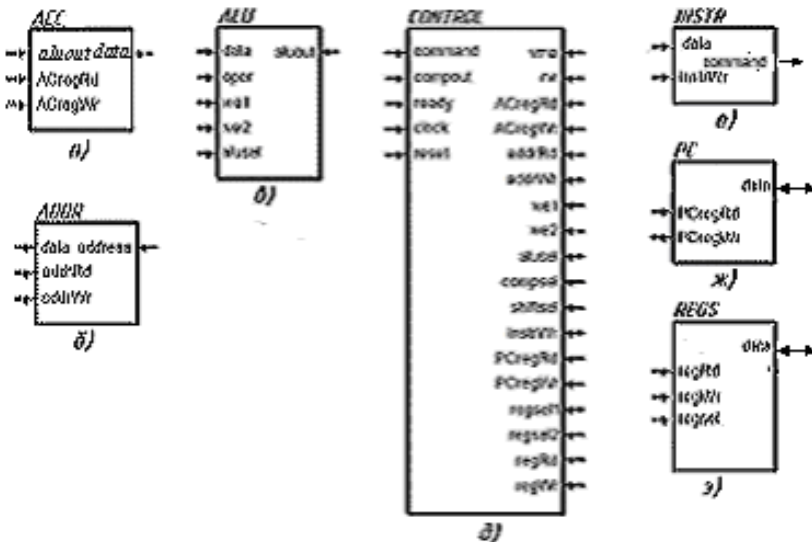
Регістр машинної команди INSTR є 16-розрядний регістр, що призначений для збереження бінарного коду чергової команди програми. Умовне

графічне позначення регістра адреси **наведене** на рис. 1.3е. Призначення портів РМК:

data	вх.	(16 біт)	загальна магістраль даних;
InstrWr	вх.	(1 біт)	сигнал запису даних до РМК;
command	вих.	(16 біт)	магістраль команди.

Лічильник команд PC (program counter) формує поточну 16-розрядну адресу машинної команди в пам'яті. Значення **PC** збільшується на одиницю при виборі кожного слова команди або завантажується за допомогою інструкції переходу. Умовне графічне позначення ЛК **наведене** на рис. 1.3ж. Призначення портів лічильника команд:

Data	вх./ вих	(16 біт)	загальна магістраль даних;
PCregRd	вх.	(1 біт)	дозвіл читання вмісту ЛК;
PCregWr	вх.	(1 біт)	сигнал запису даних до ЛК.



а – акумулятор; б – регістр адреси; в – АЛП; д – пристрій керування;
е – РМК; ж – ЛК; з – регістровий файл

Рисунок 1.3 – Архітектурні компоненти МП

Регістровий файл REGS містить програмно-доступні операційні реєстри **Reg0 – Reg7** мікропроцесора. Під час запису даних звертання до цих реєстрів виконується за індексом **regsel=№Reg** за сигналом **regWr**. Під час читання вмісту операндів – за допомогою сигналів **regsel** і **regRd**. Умовне графічне позначення реєстрового файлу наведене на рис. 1.3з.

Призначення його портів:

data	вх/ вих.	(16 біт)	загальна магістраль даних;
regsel	вх.	(3 біти)	сигнал вибору реєстра операнда;
regRd	вх.	(1 біт)	дозвіл читання вмісту реєстрової пам'яті;
regWr	вх.	(1 біт)	сигнал запису даних до реєстрової пам'яті.

На рис. 1.4 наведена логічна схема, на якій показані всі необхідні з'єднання архітектурних компонентів МП.

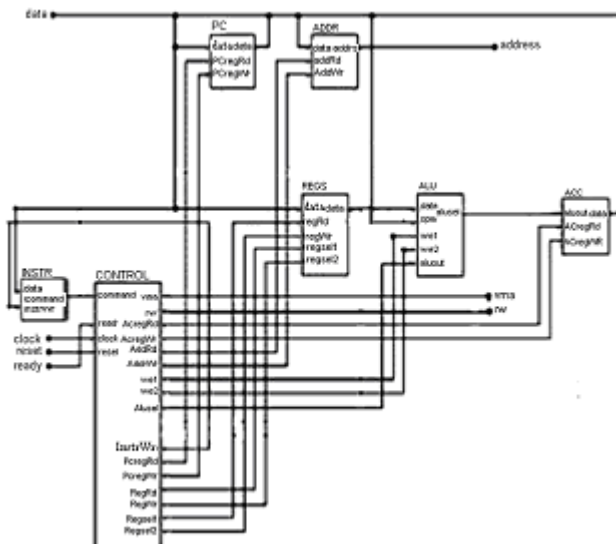


Рисунок 1.4 – Логічна схема МП

2 ПРОЕКТУВАННЯ МОДЕЛІ МІКРОПРОЦЕСОРА

2.1 Система команд мікропроцесора

Система команд МП, що проектується, наведена в табл. 1.3.

За *способом кодування* бінарні коди машинних команд можуть бути двох типів. Перший тип – команда, що складається з одного слова. Формат такої команди наведений в табл. 5. Команди другого типу складаються з двох слів (див. табл.2.1).

Таблиця 2.1- Формат команди МП першого типу

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE								DEST			SRC2			SRC1	

Таблиця 2.2 - Формат команди МП другого типу

Перше слово	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	OPCODE								DEST			SRC2			SRC1	
Друге слово	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ADDRESS / DATA															

де:

OPCODE – бітове поле (5 біт), що містить код операції;

SRC1 – бітове поле (3 біти), що містить код першого регістра джерела даних;

SRC2 – бітове поле (3 біти), що містить код другого регістра джерела даних;

DEST – бітове поле (3 біти), що містить код регістра призначення;

ADDRESS – бітове поле (16 біт), адреса переходу, якщо comprot = '1';

DATA – бітове поле (16 біт), дані (якщо використовується безпосередня адресація).

Бітові поля **SRC1**, **SRC2**, **DEST** мають розрядність 3 біти. За їх допомогою ідентифікуються номери регістрів, відповідно операнда 1, операнда 2 і результату. Наприклад, код SRC1 = "010" вказує на те, що в якості першого операнда використовується операційний регістр **Reg2**.

З наведених команд тільки команда **LOADI** складаються з двох слів.

2.2 Інформаційний тракт мікропроцесора

Керуючий пристрій і підпорядковані йому архітектурні компоненти МП створюють **операційний пристрій (ОП)**. У цьому випадку роль **керуючого автомата** ОП грає пристрій керування; решта компонентів створюють **операційний автомат**. Такий підхід спрощує проектування і полегшує розуміння процесу функціонування *пристрою обробки даних*, яким є МП (рис. 2.1).

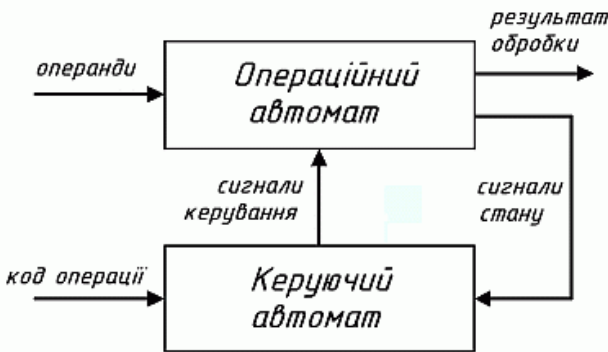


Рисунок 2.1 Функціональна схема пристрою обробки даних

Бінарний **код операції** (КОП) і **сигнали стану**, що надходять до керуючого пристрою, впливають на послідовність формування **сигналів керування**. Під впливом сигналів керування операційний автомат виконує обробку *операндів* і формує *результат обробки*.

Сукупність **мікрооперацій** (МО), що виконуються операційним автоматом, в залежності від коду машинної команди і сигналів стану, формує **інформаційний тракт МП**.

2.2.1 Команда NOP

За інструкцією **NOP** (not operation) МП не виконує ніяких перетворень даних. Часова діаграма зміни станів керуючого пристрою містить "холостий" МЦ. Формат команди наведений в табл.1.7. Час виконання інструкції NOP – 1 машинний цикл.

Таблиця 2.2 - Формат команди NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0											

Послідовність транзакцій інформаційного тракту МП під час її виконання:

- **МЦ1** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій переходить до виконання послідовності;
- **IncPC** інкремент вмісту ЛК.

2.2.2 Команда ADD

Двооперандна команда **ADD** відноситься до команд АЛП. Формат команди наведений в табл. 2.3.

Час виконання – 4 МЦ.

Таблиця 2.3- Формат двооперандної команди АЛП

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE							DEST			SRC2			SRC1		

- 1 **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій формує список задіяних операндів. Встановлюються сигнали **regsel (2:0)** = номеру задіяного регістра **SRC1**. По сигналу **regRd = '1'** дані цього регістра надходять до магістралі **data (15:0)** (перший операнд) і по сигналу **we1 = '1'** завантажуються до АЛУ.
- 2 **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** встановлюється сигнали **regsel (2:0)** = номеру регістра **SRC2**. По сигналу **regRd = '1'** дані цього регістра надходять до магістралі **data (15:0)** (другий операнд) і по сигналу **we2 = '1'** завантажуються до АЛУ.
- 3 **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попереднє встановлені сигнали і передає до АЛП сигнал **alusel (4:0)** – код функції АЛП **ADD** (бітове поле **OPCODE**), за яким виконується задана операція. Встановлюється сигнал **ACregWr = '1'** – запис даних до акумулятора.
- 4 **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попереднє встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. На вхід **regsel (2:0)** регіс-

трової пам'яті передається номер регістра-одержувача (бітове поле **DEST** бінарного коду команди). За сигналом **regWr = '1'** дані акумулятора через магістраль **data (15:0)** завантажуються за призначенням.

2.2.3 Команда PASS

Однооперандна команда **PASS** відноситься до команд АЛП. Формат команди наведений в табл. 9.

Час виконання – 3 МЦ.

Таблиця 2.3 – Формат однооперандної команди АЛП

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE							DEST						SRC1		

- 1 **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій формує список задіяних операндів. Встановлюються сигнали **regsel (2:0)** (номер задіяного регістра **SRC1**). По сигналу **regRd** дані цього регістра надходять до магістралі **data (15:0)** і по сигналу **we1 = '1'** завантажуються до АЛУ.
- 2 **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і передає до АЛП сигнал **alusel (4:0)** – код функції **PASS** (бітове поле **OPCODE**), за яким операнд не змінюється. Встановлюється сигнал **ACregWr = '1'** – запис даних до акумулятора і операнд завантажуються до акумулятора.
- 3 **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. На вхід **regsel (2:0)** регістрової пам'яті передається номер регістра-одержувача (бітове поле **DEST** бінарного коду команди). За сигналом **regWr = '1'** дані акумулятора через магістраль **data (15:0)** завантажуються за призначенням.

2.2.4 Команда INC

Однооперандна команда **INC** відноситься до команд АЛП. Формат команди наведений в табл. 2.3.

Час виконання – 3 МЦ.

Таблиця 2.3 - Формат **однооперандної** команди АЛП

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE							DEST						SRC1		

- 1 **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій формує список задіяних операндів. Встановлюються сигнали **regsel (2:0)** = номеру задіяного регістра **SRC1**. За сигналом **regRd** дані цього регістра надходять до магістралі **data (15:0)** і за сигналом **we1 = '1'** завантажуються до АЛУ.
- 2 **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і передає до АЛП сигнал **alusel (4:0)** – код функції **INC** (бітове поле **OPCODE**), за яким до операнда додається одиниця. Встановлюється сигнал **ACregWr = '1'** – запис даних до акумулятора і результат завантажуються до акумулятора.
- 3 **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попереднє встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. На вхід **regsel (2:0)** регістрової пам'яті передається номер регістра-одержувача (бітове поле **DEST** бінарного коду команди). За сигналом **regWr = '1'** дані акумулятора через магістраль **data (15:0)** завантажуються за призначенням.

2.2.5 Команда ZERO

Формат команди **ZERO** наведений в табл. 11. Час виконання – 2 МЦ. Послідовність транзакцій інформаційного тракту МП.

Таблиця 2.4 - Формат команди ZERO

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1			DEST								

- 1 **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій передає до АЛП сигнал **alusel (4:0)** = "01001" – код функції **ZERO**, за яким на виході АЛУ встановлюється '0'. По сигналу **ACregWr = '1'** дані записуються до акумулятора.
- 2 **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. На

вхід **regsel1 (2:0)** регістрової пам'яті передається номер регістра-одержувача (бітове поле **DEST** бінарного коду команди). За допомогою сигналу **regWr = '1'** дані акумулятора через магістраль **data (15:0)** завантажуються за призначенням

2.2.6 Команда STORE

Формат команди **STORE** наведений в табл. 2.5. Час виконання – 3 машинні цикли. Послідовність транзакцій інформаційного тракту МП.

Таблиця 2.5 - Формат команди STORE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0			DEST							SRC1	

- МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій встановлює сигнал **regsel** – номер регістра **DEST**. За допомогою сигналу **regRd** дані надходять на магістраль **data (15:0)**. Сигналом **addrWr** дані завантажуються до регістра адреси.
- МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і встановлює сигнал **regsel (2:0)** – номер регістра **SRC1**. За допомогою сигналу **regRd** дані першого операнда надходять на магістраль **data (15:0)**. Керуючий пристрій скидає попередньо встановлені сигнали і формує сигнали **rw = '1'** – операція запису в пам'ять і **vma = '1'** – підтвердження адреси. Цими сигналами мікропроцесор починає цикл запису в зовнішню пам'ять.
- МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали.

2.2.7 Команда LOADI

Формат команди **LOADI** наведений в табл. 2.6. Час виконання – 5 машинних циклів.

Таблиця 2.6 - Формат команди LOADI

Перше слово	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	0			DEST								
Друге слово	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DATA															

Послідовність транзакцій інформаційного тракту МП.

1. **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій встановлює сигнал **PCregRd** = '1' – читання вмісту ЛК. Данні лічильника команд надходять на магістраль **data (15:0)** (перший операнд). Сигналом **we1** операнд завантажується до АЛП.
2. **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і передає до АЛП сигнал **alusel (4:0)** = "00110" – код функції **INC**. До регістра зсуву передається сигнал **shiftsel (4:0)** = "10001" – код функції **PASS**. Встановлюється сигнал **ACregWr** = '1' – запис даних до акумулятора.
3. **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і формує сигнал **ACregRd** = '1' – читання вмісту акумулятора. Дані акумулятора виставляються на магістраль **data (15:0)**. Сигналами **PCregWr** і **addrWr** вміст акумулятора завантажується до лічильника команд і регістра адреси відповідно.
4. **МЦ4:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і формує сигнали **rw** = '0' – операція читання пам'яті і **vma** = '1' – підтвердження адреси. Цими сигналами мікропроцесор починає цикл читання зовнішньої пам'яті.
5. **МЦ5:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій перевіряє стан сигналу **ready** – підтвердження даних на магістралі **data (15:0)**. Якщо **ready** = '1' (для асинхронних МП), керуючий пристрій скидає попередньо встановлені сигнали і передає номер регістра-одержувача (бітове поле **DEST** бінарного коду команди) на вхід **regsel** регістрової пам'яті. За допомогою сигналу **regWr** = '1' дані магістралі **data (15:0)** завантажуються за призначенням.

2.3 Тракт керування мікропроцесора

Сукупність мікрооперацій (МО), що виконуються керуючим автоматом для забезпечення нормального функціонування МП формує **тракт керування МП**.

Крім дій інформаційного тракту мікропроцесора, які обумовлені поточною машинною командою, керуючий пристрій забезпечує реалізацію операцій тракту керування. Тракт керування МП реалізують послідовності МО:

reset – скидання і початок роботи МП;

execute – читування і декодування машинної команди;

oper – виконання машинної команди;

IncPC – інкремент ЛК;

LOADPC – завантаження нової адреси до ЛК.

У керуючому пристрої мікропроцесора використовується **схемний принцип керування**. *Граф переходів* керуючого автомата МП наведений на рис. 2.1.

Після надходження сигналу **reset** = '1' керуючий пристрій автоматично виконує послідовність операцій **reset**. До лічильника команд та адресного регістра МП завантажується адреса початку програми, що зберігається у зовнішній пам'яті – 0000h.

Для завантаження поточної машинної команди до РМК та її декодування використовується послідовність операцій **execute**.

Після виконання більшості команд інформаційного тракту, скидання попередніх сигналів МП значення лічильника команд збільшується на одиницю (послідовність **IncPC**).

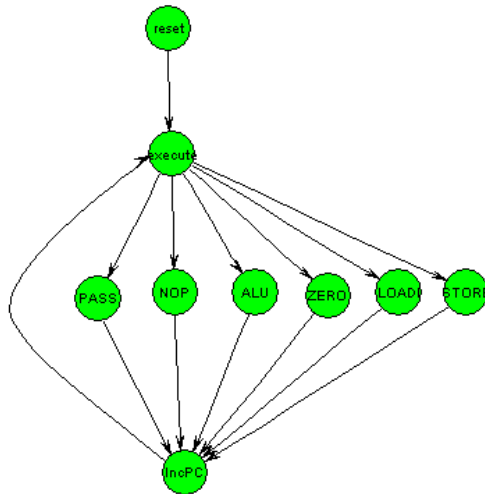


Рисунок 2.1- Граф переходів керуючого автомата МП

2.3.1 Скидання і початок роботи мікропроцесора

Час виконання послідовності **reset** – 2 машинних цикли.

Трансакції тракту керування МП:

1. **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій передає до АЛП сигнал **alusel** (4:0) = "01001" – код

функції **ZERO**, за яким АЛП обнуляється. Встановлюється сигнал **ACregWr = '1'** – запис даних до акумулятора.

2. **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. За допомогою сигналів **addrWr = '1'**, **PCregWr = '1'** дані акумулятора через магістраль **data (15:0)** завантажуються до регістра адреси і ЛК.

2.3.2 Зчитування та декодування команди

Час виконання послідовності **execute** – 4 машинних цикли:

1. **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і формує сигнали **rw = '0'** – операція читання пам'яті і **vma = '1'** – підтвердження адреси. Цими сигналами мікропроцесор починає цикл читання зовнішньої пам'яті
2. **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій перевіряє стан сигналу **ready** – підтвердження даних на магістралі **data (15:0)** при наявності сигналу **vma = '1'**. Якщо **ready = '0'**, керуючий пристрій чекає поки пам'ять не виставить дані на шину **data**. Якщо **ready = '1'**, керуючий пристрій скидає попередньо встановлені сигнали і за допомогою сигналу **instrWr** дані магістралі **data (15:0)** завантажуються до регістра машинної команди. Через магістраль **data (15:0)** машинна команда із РМК надходить до керуючого пристроя.
3. **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** скидає попередні сигнали (**vma = '0'**, **instrWr = '0'**) керуючий пристрій виділяє з коду команди **код операції** і **коди номерів регістрів** operandів та результату, за якими встановлюються номери відповідних регістрів.
4. **МЦ4:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій виконує її декодування: по виділеним кодам команд встановлюються назви операцій. Після чого настає етап виконання команди.

2.3.3 Інкремент лічильника команд

Час виконання послідовності **IncPC** – 3 машинних цикли. Транзакції тракту керування МП:

1. **МЦ1:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій встановлює сигнал **PCregRd** – читання лічильника команд. Дані надходять на магістраль **data (15:0)** (перший операнд). Встановлюється сигнал **we1 = '1'** – завантаження операнда до арифметично-логічного пристрою.
2. **МЦ2:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і передає до АЛП сигнал **alusel (4:0) = "00110"** – код функції **INC**, за яким до операнда додається одиниця. Встановлюється сигнал **ACregWr = '1'** – запис результату до акумулятора.
3. **МЦ3:** після надходження зростаючого фронту тактового сигналу **clock** керуючий пристрій скидає попередньо встановлені сигнали і встановлює сигнал **ACregRd = '1'** – читання даних акумулятора. Дані акумулятора виставляються на магістраль **data (15:0)**. Сигналами **addrWr = '1'**, **PCregWr = '1'** вміст акумулятора завантажується до регістра адреси і лічильника команд.

2.4 Моделі мікропроцесора

Проект з ім'ям **comp** повинен містити 10 файлів VHDL-коду:

mem.vhd	– модель зовнішньої пам'яті для МП;
cpulib.vhd	– глобальні визначення для проекту;
clk_gen.vhd	– модель генератора тактових сигналів;
top_cpu.vhd	– модель верхнього рівня проекту;
reg.vhd	– модель регістра МП;
trireg.vhd	– модель регістра з трьома станами;
regarray.vhd	– модель масиву регістрів МП;
alu.vhd	– модель АЛП МП;
control.vhd	– модель керуючого пристрою МП;
cpu.vhd	– структурна модель МП на рівні компонентів.

Із них три (**mem.vhd**, **clk_den.vhd**, **top_cpu.vhd**) розроблені в лаб.3, а інші, що входять до складу МП, треба ще розробити.

2.4.1 Модель регістра

Модель регістра МП міститься у файлі **reg.vhd** (листинг 2). Вона використовується як модель РМК (INSTR) і РгА (ADDR) МП.

Інтерфейс регістра (сутність **reg**) містить наступні сигнали:

a	вх.	(16 біт)	вхідні дані регістра (16-розрядний бінарний вектор);
clk	вх.	(1 біт)	сигнал запису даних;

q вих. (16 біт) вихідні дані.

Процес **RUN** в архітектурному тілі моделі регістра аналізує наявність позитивного фронту на вході **clk** і у разі його наявності здійснює реєстрацію вхідних даних.

Лістинг 2– Файл reg.vhd

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpulib.ALL;
ENTITY reg IS
PORT
(
  a: IN STD_LOGIC_VECTOR (15 DOWNTO 0); -- a
  clk: IN STD_LOGIC; -- clk
  q: OUT STD_LOGIC_VECTOR(15 DOWNTO 0) -- q
);
END reg;
ARCHITECTURE reg of reg IS
SIGNAL tmp: STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
RUN: PROCESS (clk)
BEGIN
  IF clk'event AND clk = '1' THEN
    tmp <= a;
  END IF;
END PROCESS;
q <= tmp;
END reg;

```

2.4.2 Модель регістра з трьома станами

Файл **trireg.vhd** (лістинг 3) містить модель регістра з трьома станами. Вона використовується як модель акумулятора **ACC** і програмного лічильника **PC** МП.

Інтерфейс регістра з трьома станами (сутність **trireg**) містить наступні сигнали:

a	вх.	(16 біт)	вхідні дані регістра (16-розрядний бінарний вектор);
clk	вх.	(1 біт)	сигнал запису даних;
En	вх.	(1 біт)	сигнал фіксації даних на виході регістра;
Q	вих.	(16 біт)	вихідні дані.

Архітектурне тіло моделі містить два процеси. Процес **RUN1** аналізує наявність позитивного фронту на вході **clk** і у разі його наявності завантажує вхідні данні. Процес **RUN2** аналізує стан входу **en**. У випадку, якщо **en = '1'**, дані зберігаються і фіксуються на виході регістра; якщо **en ≠ '1'**, то вихід регістра переходить в третій стан.

Лістинг 3– Файл `trireg.vhd`

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY trireg IS
  PORT
    (
      a : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- a _____|
      clk : IN STD_LOGIC; -- clk _____|
      en : IN STD_LOGIC; -- en _____|
      q : out STD_LOGIC_VECTOR(15 DOWNTO 0) -- | _____| q
    );
END ENTITY trireg;
ARCHITECTURE trireg of trireg IS
  SIGNAL res : std_logic_vector(15 DOWNTO 0);
  BEGIN
    RUN1: PROCESS (a, clk)
      BEGIN
        IF clk'event AND clk = '1' THEN
          res <= a;
        END IF;
      END PROCESS;
    RUN2: PROCESS (res, en)
      BEGIN
        IF en = '1' THEN
          q <= res;
        ELSE
          q <= "ZZZZZZZZZZZZZZZZ";
        END IF;
      END PROCESS;
    END ARCHITECTURE trireg;

```

Часові діаграми, які отримані під час функціонального моделювання компонента *trireg*, наведені на рис.2.2.

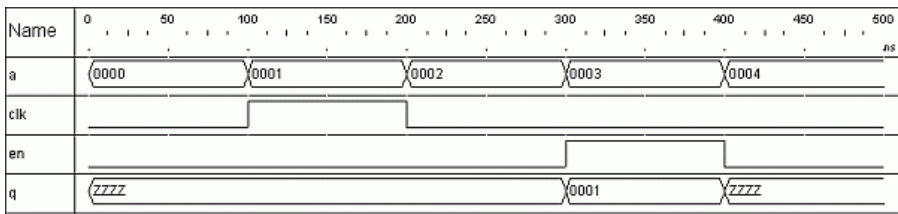


Рисунок 2.2 - Моделювання регістра з трьома станами

2.4.3 Модель регістрової пам'яті МП

Модель регістрової пам'яті (масиву з восьми регістрів *Reg0...Reg7*) знаходиться у файлі **regarray.vhd** (лістинг 4).

Стан регістрової пам'яті описується сигналом **ramdata**, що є масивом з восьми 16-розрядних слів. Інтерфейс регістрової пам'яті (сутність *regarray*) містить наступні сигнали:

data	вх.	(16 біт)	вхідні дані;
sel	вх.	t_reg	вибір регістра;
En	вх.	(1 біт)	сигнал фіксації даних на виході регістра;
clk	вх.	(1 біт)	сигнал запису даних;
Q	вих.	(16 біт)	вихідні дані.

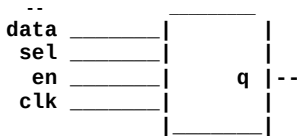
Архітектурне тіло моделі регістрової пам'яті містить дві частини процесу. Перша частина процесу аналізує наявність позитивного фронту на вході **clk** і у разі його наявності завантажує вхідні дані. Друга частина процесу аналізує стан входу **en**. У випадку, якщо **en** = '1', дані, що зберігаються, фіксуються на виході регістрової пам'яті; якщо **en** $\neq$ '1', то вихід регістра встановлюється у третій стан.

Лістинг 4 – Файл regarray.vhd

```

library IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE work.cpulib.ALL;
ENTITY regarray IS
PORT
(
    data:IN STD_LOGIC_VECTOR(15 DOWNT0 );-- data
    sel : IN    t_reg;                -- sel
    en  : IN    STD_LOGIC;            -- en
    clk : IN    STD_LOGIC;            -- clk
    q:OUT STD_LOGIC_VECTOR(15 DOWNT0 ) --
);
END regarray;
ARCHITECTURE regarray of regarray IS
TYPE t_ram IS ARRAY (0 TO 7) STD_LOGIC_VECTOR (15 downto 0);
SIGNAL ramdata : t_ram :=
(
    "0000000000000000", -- perictp R0
    "0000000000000000", -- perictp R1
    "0000000000000000", -- perictp R2
    "0000000000000000", -- perictp R3
    "0000000000000000", -- perictp R4
    "0000000000000000", -- perictp R5
    "0000000000000000", -- perictp R6

```



```

"0000000000000000"      -- perictp R7
);
BEGIN
PROCESS (clk, en)
BEGIN
    IF clk'event AND clk = '1' THEN
        ramdata (sel) <= data;
    END IF;
    IF en'event AND en = '1' THEN
        q <= ramdata (sel);
    ELSE
        q <= "ZZZZZZZZZZZZZZZZ";
    END IF;
END PROCESS;
END regarray;

```

2.4.4 Пакет глобальних визначень проекту

Файл **cpulib.vhd** містить типи даних проекту **comp** (лістинг 5).

Лістинг 5– Файл cpulib.vhd

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
PACKAGE cpulib IS
TYPE state_type IS      -- стани керуючого автомата
(
    reset1, reset2,      -- послідовність виконання команди reset
    execute1,execute2,execute3,execute4, -- послідовність декодування команди
    opNOP1,              -- послідовність виконання команди NOP
    opPASS1, opPASS2, opPASS3,      -- "-"- "-"- команди PASS
    opADD1, opADD2, opADD3, opADD4,  -- "-"- "-"- команди ADD
    opLOADI1,opLOADI2,opLOADI3,opLOADI4,opLOADI, -- "-"- команди LOADI
    opSTORE1, opSTORE2 , opSTORE3,   -- "-"- "-"- команди STORE
    opIncPC1, opIncPC2, opIncPC3,     -- "-"- "-"- команди IncPC
    opINC1, opINC2, opINC3,           -- "-"- "-"- команди INC
    opZERO1, opZERO2                 -- "-"- "-"- команди ZERO
);
TYPE op_type IS      -- операції мікропроцесора (див табл.1.5)
(
    opNOP,            -- немає операції
    opADD,            -- складання
    opPASS,          -- передача без змін
    opINC,            -- інкремент
    opIncPC,          -- інкремент лічильника PC
    opLOADI,          -- завантаження регістра
    opSTORE,          -- завантаження даних до пам'яті
    opZERO            -- нуль
);

```

```

SUBTYPE t_reg    IS INTEGER RANGE 0 TO 7;  -- операційні регістри
SUBTYPE t_alu    IS op_type RANGE opNOP to opZERO; -- команди АЛУ
END cpulib;

```

Тип даних *state_type* задає всі можливі стани керуючого пристрою МП. Тип даних *op_type* дозволяє задати вид операції арифметично-логічного пристрою (сутність *alu*). За допомогою підтипу даних *t_reg* здійснюється вибір потрібного регістра в масиві **REGS** з 8 регістрів.

2.4.5 Модель АЛП

Модель АЛП МП міститься у файлі **alu.vhd** (листинг 6). Він обробляє керуючі сигнали, що належать до алфавіту функцій АЛП $O_{alu} = \{ZERO, INC, PASS, ADD\}$.

Інтерфейс АЛП (сутність *alu*) містить наступні сигнали:

data	вх.	(16 біт)	дані першого операнда;
oper	вх.	(16 біт)	дані другого операнда;
we1	вх.	(1 біт)	сигнал завантаження даних першого операнда;
we2	вх.	(1 біт)	сигнал завантаження даних другого операнда;
func	вх.	op_type	сигнал вибору функцій АЛП;
result	вих.	(16 біт)	магістраль результату АЛП.

Архітектурне тіло моделі містить два процеси:

- процес **RUN1**, в якому за сигналом **we1** в АЛП подається перший операнд **op1**, а за сигналом **we2** - другий операнд **op2**;
- процес **RUN2**, в якому послідовний оператор **CASE** аналізує сигнал **func** і відповідно до його значення виконує необхідну операцію АЛП.

Лістинг 6 – Файл alu.vhd

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE work.cpulib.ALL;
ENTITY calculate IS
PORT
(
    data : IN STD_LOGIC_VECTOR (15 DOWNT0 0); -- _____|data |
    oper : IN STD_LOGIC_VECTOR (15 DOWNT0 0); -- _____|oper |
    we1  : IN STD_LOGIC; -- _____|we1 |
    we2  : IN STD_LOGIC; -- _____|we2 |
    func : IN op_type; -- _____|func |
    result : OUT STD_LOGIC_VECTOR (15 DOWNT0 0) -- _____| result|_____
);
END calculate;
ARCHITECTURE calculate of calculate IS

```

```

SIGNAL op1 : STD_LOGIC_VECTOR (15 DOWNT0 0) := "ZZZZZZZZZZZZZZZZ";
SIGNAL op2 : STD_LOGIC_VECTOR (15 DOWNT0 0) := "ZZZZZZZZZZZZZZZZ";
BEGIN
RUN1: PROCESS (we1, we2)
  BEGIN
    IF we1'event AND we1 = '1' THEN
      op1 <= data;
    ELSIF we2'event AND we2 = '1' THEN
      op2 <= oper;
    END IF;
  END PROCESS;
RUN2: PROCESS (func)
  BEGIN
    CASE func IS
      WHEN opZERO => -- Нуль
        result <= "0000000000000000";
      WHEN opINC => -- інкремент
        result <= op1 + "0000000000000001";
      WHEN opPASS => -- Операція PASS
        result <= op1;
      WHEN opADD => -- Операція складання
        result <= op2 + op1;
      WHEN OTHERS => NULL;
    END CASE;
  END PROCESS;
END calculate;

```

2.4.6 Складання моделі керуючого пристрою

Сукупність послідовностей виконання машинних операцій по машинних циклах, які розглядались раніше, складають *алфавіт станів* керуючого пристрою МП $A = \{a_1, a_2, \dots, a_n\}$. В таблиці 2.6 наведений опис усіх станів керуючого пристрою МП.

Таблиця 2.6 - Таблиця переходів керуючого автомата

Стан- передавач	Вихідні сигнали	Стан- приймач	ЛУ	Послідовність вико- нання операцій
reset 1	alusel<= opzero; ACregWr<= '1'after 1ns;	reset 2	1	0 -> ALU-> [Ac]
reset 2	ACregRd <= '1'; PCregWr <= '1'after 1ns; AddrWr<= '1'after 1ns;	execute 1	1	[Ac] -> data-> [PC] , data-> [Adr]
execute 1	AddrWr<= '0'; PCregWr <= '0'; ACregRd <= '0'; Rw <= '0'; vma<='1';	execute 2	1	[Adr] ->address->mem

execute 2	vma<='1';	execute 2	ready='0'	
	InstrWr<='1';	execute 3	ready='1'	mem->data->[Instr] ->Contr
execute 3	vma<='0'; InstrWr<='0'; operbit , srcregbit1, srcregbit2, dstregbit	execute 4	1	Contr: operbit, srcregbit1, srcregbit2, dstregbit
execute 4	oper, source1, source2, dest	opNOP1, ... opSTORE1	oper="00000" ... oper="11000"	Contr: oper, №Rsrc1, №Rsrc1, №Rdest
opNOP1		opIncPC1	1	
opPASS1	regsel<= source1; regRd<='1'; we1_alu <= '1'after 1ns;	opPASS2	1	[Rsrc1] ->data->ALU
opPASS2	alusel<= opPASS; ACregWr<= '1'after 1ns;	opPASS3	1	ALU: [Rsrc1] -> [Ac]
opPASS3	regsel<= dest; ACregRd<='1'; regWr <= '1'after 1ns;	opIncPC1	1	[Ac] -> data-> [Rdest]
opZERO1	alusel<= opZERO; ACregWr<= '1'after 1ns;	opZERO2	1	ALU: 0 -> [Ac]
opZERO2	regsel<= dest; regWr <= '1'after 1ns;	opIncPC1	1	[Ac] -> data-> [Rdest]
Стан- передавач	Вихідні сигнали	Стан- приймач	ЛІУ	Послідовність вико- нання операцій
opINC1	regsel<= source1; regRd<='1'; we1_alu <= '1'after 1ns;	opINC2	1	[Rsrc1] ->data->ALU
opINC2	alusel<=opINC; ACregWr<= '1'after 1ns;	opINC3	1	ALU: [Rsrc1] +1-> [Ac]
opINC3	ACregRd<='1'; regsel<= dest; regWr <= '1'after 1ns;	opIncPC1	1	[Ac] -> data-> [Rdest]
opIncPC1	PCregRd<='1'; we1_alu <= '1'after 1ns;	opIncPC2	1	[PC] -> data-> ALU
opIncPC2	alusel<= opINC; ACregWr<= '1'after 1ns;	opIncPC3	1	ALU: [PC] +1-> [Ac]
opIncPC3	ACregRd <= '1'; PCregWr<='1'after1ns; addrWr <= '1'after 1ns;	excecute1	1	[Ac] -> data-> [PC], data-> [Adr]
opALU1 (ADD)	regsel<= source1; regRd<='1'; we1_alu <= '1'after 1ns;	opALU2	1	[Rsrc1] ->data->ALU

opALU2	regsel<= source2; regRd<='1'; we2_alu <= '1'after 1ns;	opALU3	1	[Rsrc2] ->data->ALU
opALU3	alusel<=opADD; ACregWr<= '1'after 1ns;	opALU4	1	ALU: [Rsrc1]+ [Rsrc2] -> [Ac]
opALU4	ACregRd<='1'; regsel<= dest; regWr <= '1'after 1ns;	opIncPC1	1	[Ac] -> data-> [Rdest]
opLOADI1	PCregRd<='1'; we1_alu <= '1'after 1ns	opLOADI2	1	[PC] -> data-> ALU
opLOADI2	alusel<= INC; ACregWr<= '1'after 1ns;	opLOADI3	1	ALU: [PC] +1-> [Ac]
opLOADI3	ACregRd <= '1'; PCregWr<= '1'after 1ns; addrWr <= '1'after 1ns;	opLOADI4	1	[Ac] -> data-> [PC], data-> [Adr]
opLOADI4	rw <= '0'; vma<='1';	opLOADI5	1	[Adr] ->address->mem
opLOADI5	vma<='1';	opLOADI5	ready='0'	
	Regsel<= dest; regWr <= '1'after 1ns; vma<='0'after 1ns;	opIncPC1	ready='1',	mem(address)->data ->[Rdest]
opSTORE1	Regsel<= dest; regRd<='1'; addrWr <= '1'after 1ns;	OpSTORE2	1	[Rdest] ->data-> [Adr]
opSTORE2	Regsel<= source1; regRd<='1'; addrRd <= '1'; rw <= '1'; vma<='1';	opSTORE3	1	[Rsrc1] ->data, [Adr] ->address->mem, data -> mem(address)
opSTORE3	rw <= '0'; vma<='0'	opIncPC1	1	

В таблиці 2.6 не показані скидання попередніх сигналів у відповідності до МЦ виконання команд.

Модель керуючого автомата міститься у файлі **control.vhd** (лістинг 7). Архітектурне тіло моделі керуючого автомата містить один процес. Список чутливості процесу показує, що цей процес активізується в наступних випадках:

- відбулася зміна стану керуючого пристроя (сигнал **state**);
- поступила для виконання команда скидання (сигнал **reset**).

Активізуючись, даний процес перш за все виконує ініціалізацію вихідних керуючих сигналів і аналізує вхід **reset**:

- у разі надходження сигналу скидання пристрій повертається в стан **reset**;

- якщо ж сигналу скидання немає, то процес аналізує наявність позитивного фронту на вході синхросигналу **clk** і, якщо позитивний фронт в наявності, здійснюється перехід в наступний стан керуючого автомата.

Потім за допомогою оператора **CASE** аналізується поточний стан, і залежно від результату аналізу виконується один з ланцюжків переходів між станами пристроя (разом з операціями, пов'язаними з кожним з цих станів): **reset, execute, NOP, PASS, ZERO, STORE, LOADi, IncPC, ADD**.

Лістинг 7– Файл control.vhd

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE work.cpulib.ALL;
entity core is
PORT
  (
    reset      : IN   STD_LOGIC;
    clock      : IN   STD_LOGIC;
    command    : IN   STD_LOGIC_VECTOR(15 DOWNTO 0);
    ready      : IN   STD_LOGIC;
    aluSel     : OUT  op_type;
    regSel     : OUT  t_reg;
    PCregWr    : OUT  STD_LOGIC;
    PCregRd    : OUT  STD_LOGIC;
    addrWr     : OUT  STD_LOGIC;
    AcregWr    : OUT  STD_LOGIC;
    AcregRd    : OUT  STD_LOGIC;
    instrWr    : OUT  STD_LOGIC;
    regRd      : OUT  STD_LOGIC;
    regWr      : OUT  STD_LOGIC;
    we1_alu    : OUT  STD_LOGIC;
    we2_alu    : OUT  STD_LOGIC;
    rw         : OUT  STD_LOGIC;
    vma        : OUT  STD_LOGIC
  );
END;
ARCHITECTURE core of core IS
SIGNAL state      : state_type;           -- стан автомата
SIGNAL oper       : op_type;              -- операція
SIGNAL source1    : t_reg;               -- джерело / операнд 1
SIGNAL source2    : t_reg;              -- джерело / операнд 2
SIGNAL dest       : t_reg;              -- одержувач
SIGNAL operbit    : STD_LOGIC_VECTOR(4 DOWNTO 0); -- бітове поле операції
SIGNAL srcregbit1 : STD_LOGIC_VECTOR(2 DOWNTO 0); -- біт поле джерела / операнда 1
SIGNAL srcregbit2 : STD_LOGIC_VECTOR(2 DOWNTO 0); -- біт поле джерела / операнда 2

```

```

SIGNAL dstregbit : STD_LOGIC_VECTOR (2 DOWNTO 0);-- бітове поле одержувача
BEGIN
PROCESS (clock, reset)
    BEGIN
we1_alu <= '0';      -- ініціалізація сигналів
we2_alu <= '0';
aluSel  <= opNOP;
ACregWr <= '0';
ACregRd <= '0';
regRd   <= '0';
regWr   <= '0';
regSel  <= 0;
PCregWr <= '0';
PCregRd <= '0';
rw      <= '0';
IF reset='1' THEN
    state <= reset1;
ELSIF clock'event and clock = '1' THEN
    CASE state is
        -----
        -- СКИДАННЯ
        WHEN reset1 =>
            -- Такт 1:
            aluSel <= opZERO;      -- результат АЛП - нуль
            ACregWr <= '1' AFTER 1 ns; -- запис до акумулятора
            state <= reset2;
        WHEN reset2 =>
            -- Такт 2:
            ACregWr <= '0';
            ACregRd <= '1';      -- Читання акумулятора
            PCregWr <= '1' AFTER 1 ns; --Запис до лічильника команд
            addrWr <= '1' AFTER 1 ns; --Запис до регістра адреси
            state <= execute1;
        -----
        WHEN execute1=>
            -- Такт 1:
            ACregRd <= '0';
            addrWr <= '0';
            PCregWr <= '0';
            rw <= '0';
            vma <= '1';      -- Фіксація адреси
            state <= execute2;
        WHEN execute2=>
            -- Такт 2:
            vma <= '1';
            IF ready = '1' THEN
                instrWr <= '1';      -- Запис до регістра команд
                state <= execute3;
            ELSE
                state <= execute2;      -- Очікування підтвердження
            END IF;
            -- ДЕШИФРАЦІЯ КОМАНДИ
        WHEN execute3 =>
            -- Такт 3:
            vma <= '0';      -- Скидання сигналу vma
            instrWr <= '0';      -- Скидання сигналу INSTRWR

operbit <= std_logic_vector'(command(15) &command(14) &command(13))

```

```

&command(12) &command(11));
srcregbit1 <= std_logic_vector'(command(2)&command(1)&command(0));
srcregbit2 <= std_logic_vector'(command(5)&command(4)&command(3));
dstregbit  <= std_logic_vector'(command(8)&command(7)&command(6));
state      <= execute4;

WHEN execute4 =>
    IF operbit = std_logic_vector'("00000") THEN
        oper <= opNOP;
        state <= opNOP1;
    ELSIF operbit = std_logic_vector'("00101") THEN
        oper <= opADD;
        state <= opADD1;
    ELSIF operbit = std_logic_vector'("00110") THEN
        oper <= opINC;
        state <= opINC1;

    ELSIF operbit = std_logic_vector'("01001") THEN
        oper <= opZERO;
        state <= opZERO1;

    ELSIF operbit = std_logic_vector'("10001") THEN
        oper <= opPASS;
        state <= opPASS1;

    ELSIF operbit = std_logic_vector'("10110") THEN
        oper <= opLOADI;
        state <= opLOADI1;

    ELSIF operbit = std_logic_vector'("11000") THEN
        oper<=opSTORE;
        state <= opSTORE1;
    ELSE
        state <= reset1;
    END IF;

source1 <= CONV_INTEGER (srcregbit1); -- номер регістра-джерела 1
source2<= CONV_INTEGER (srcregbit2); -- номер регістра-джерела 2
dest    <= CONV_INTEGER (dstregbit); -- номер регістра-одержувача
-----
-- КОМАНДА NOP
WHEN opNOP1    =>
    state <= opIncPC1;
-----
-- КОМАНДА PASS
WHEN opPASS1=>
    regsel <= source1;
    regRd  <= '1';
    we1_alu <= '1' AFTER 1 ns;
    state <= opPASS2;
WHEN opPASS2=>
    regRd <= '0';

```

```

we1_alu    <= '0';
we2_alu    <= '0';
aluSel     <= opPASS;           -- Операція aluPASS
ACregWr     <= '1' AFTER 1 ns;  -- Запис до акумулятора
state <= opPASS3;
WHEN opPASS3=>                  -- Такт 3:
    ACregWr <= '0';
    regsel  <= dest;            -- Вибір регістра-одержувача
    ACregRd <= '1';            -- Читання файлу регістрів
    regWr   <= '1' AFTER 1 ns;  -- Запис до файлу регістрів
    state <= opIncPC1;

```

```

-- КОМАНДА ZERO
WHEN opZERO1=>                  -- Такт 1:
    aluSel  <= opZERO;         -- Операція aluZERO
    ACregWr <= '1' AFTER 1 ns;  -- Запис до акумулятора
    state <= opZERO2;
WHEN opZERO2=>                  -- Такт 2:
    ACregWr <= '0';
    regsel  <= dest;           -- Вибір регістра-одержувача
    ACregRd <= '1';           -- Читання акумулятора
    regWr   <= '1' AFTER 1 ns; -- Запис до файлу регістрів
    state <= opIncPC1;

```

```

-- КОМАНДА LOADI
WHEN opLOADI1 =>                -- Такт 1:
    PCregRd <= '1';             -- Читання лічильника команд
    we1_alu <= '1' AFTER 1 ns;  -- Запис першого операнда ALU
    state <= opLOADI2;
WHEN opLOADI2 =>                -- Такт 2:
    PCregRd <= '0';
    we1_alu <= '0';
    aluSel  <= opINC;           -- Операція інкременту
    ACregWr <= '1' AFTER 1 ns;  -- Запис до акумулятора
    state <= opLOADI3;
WHEN opLOADI3 =>                -- Такт 3:
    ACregWr <= '0';
    ACregRd <= '1';            -- Читання акумулятора
    PCregWr <= '1' AFTER 1 ns;  -- Запис до регістра PC
    addrWr  <= '1' AFTER 1 ns;  -- Запис до регістра адреси
    state <= opLOADI4;
WHEN opLOADI4 =>                -- Такт 4:
    PCregWr <= '0';
    ACregRd <= '0';
    addrWr  <= '0';
    rw      <= '0';
    vma     <= '1';            -- Фіксація адреси
    state <= opLOADI5;
WHEN opLOADI5 =>                -- Такт 5:
    vma <= '1';

```

```

IF ready = '1' THEN
  regsel    <= dest;
  regWr     <= '1';
  vma       <= '0' AFTER 1 ns;
  state <= opIncPC1;
ELSE
  state <= opLOADI5;      -- Очікування підтвердження
END IF;

```

```

-----
-- КОМАНДА INC
WHEN opINC1    =>      -- Такт 1:
  regsel    <= source1;  -- Вибір регістра-джерела 1
  regRd     <= '1';      -- Читання файлу регістрів
  we1_alu   <= '1' AFTER 1 ns; -- Запис першого операнда АЛП
  state <= opINC2;
WHEN opINC2    =>      -- Такт 2:
  regRd     <= '0';
  we1_alu   <= '0';
  we2_alu   <= '0';
  aluSel    <= opINC;     -- Операція INC
  AcregWr   <= '1' AFTER 1 ns; -- Запис до акумулятора
  state <= opINC3;
WHEN opINC3    =>      -- Такт 3:
  AcregWr   <= '0';
  AcregRd   <= '1';      -- Читання акумулятора
  regsel    <= dest;     -- Вибір регістра-одержувача
  regWr     <= '1' AFTER 1 ns; -- Запис до файлу регістрів
  state <= opIncPC1;

```

```

-----
-- КОМАНДА АЛП
WHEN opADD1    =>      -- Такт 1:
  regsel    <= source1;  -- Вибір регістра-джерела 1
  regRd     <= '1';      -- Читання файлу регістрів
  we1_alu   <= '1' AFTER 1 ns; -- Запис першого операнда АЛП
  state <= opADD2;
WHEN opADD2    =>      -- Такт 2:
  we1_alu   <= '0';
  regsel    <= source2;  -- Вибір регістра-джерела 2
  regRd     <= '1';      -- Читання файлу регістрів
  we2_alu   <= '1' AFTER 1 ns; -- Запис другого операнда АЛП
  state <= opADD3;
WHEN opADD3    =>      -- Такт 3:
  we2_alu   <= '0';
  regRd     <= '0';
  aluSel    <= opADD;     -- Операція АЛП
  AcregWr   <= '1' AFTER 1 ns; -- Запис до акумулятора
  state <= opADD4;
WHEN opADD4    =>      -- Такт 4:
  AcregWr   <= '0';
  regsel    <= dest;     -- Вибір регістра-одержувача
  AcregRd   <= '1';      -- Читання акумулятора
  regWr     <= '1' AFTER 1 ns; -- Запис до файлу регістрів
  state <= opIncPC1;

```

```

-----
                                --КОМАНДА STORE
WHEN opSTORE1 =>                                -- Такт 1:
    regsel    <= dest;                            -- Вибір регістра-одержувача
    regRd     <= '1';                            -- Читання файлу регістрів
    addrWr    <= '1' AFTER 1 ns;-- Запис до регістру адреси
    state     <= opSTORE2;
WHEN opSTORE2 =>                                -- Такт 2:
    regsel    <= source1;
    regRd     <= '1';                            -- Читання файлу регістрів
    addrWr    <= '0';
    rw        <= '1';                            -- Запис до пам'яті
    vma       <= '1';                            -- Фіксація адреси
    state     <= opSTORE3;
WHEN opSTORE3 =>                                -- Такт 3:
    rw        <= '0';
    vma       <= '0' AFTER 1 ns;
    state     <= opIncPC1;
-----

                                -- ІНКРЕМЕНТ ЛІЧИЛЬНИКА КОМАНД IncPC
WHEN opIncPC1=>                                -- Такт 1:
    PCregWr   <= '0';
    AcregRd   <= '0';
    regWr     <= '0';
    PCregRd   <= '1';                            -- Читання лічильника команд
    we1_alu   <= '1' AFTER 1 ns;-- Запис першого операнда АЛП
    state     <= opIncPC2;
WHEN opIncPC2=>                                -- Такт 2:
    PCregRd   <= '0';
    we1_alu   <= '0';
    aluSel    <= opINC;                            -- Операція інкременту
    AcregWr   <= '1' AFTER 1 ns;-- Запис до акумулятора
    state     <= opIncPC3;
WHEN opIncPC3=>                                -- Такт 3:
    AcregWr   <= '0';
    AcregRd   <= '1';                            -- Читання акумулятора
    PCregWr   <= '1' AFTER 1 ns;-- Запис до регістра команд
    addrWr    <= '1' AFTER 1 ns;-- Запис до регістра адреси
    state     <= execute1;
-----

    WHEN OTHERS =>
        state <= reset1;
END CASE;
END IF;
END PROCESS;
END core;

```

2.4.7 Модель МП

Модель мікропроцесора складається з наступних компонентів:

- regarray** – модель регістрової пам'яті (масиву регістрів Reg0...Reg7);
- reg** – модель регістра команди і регістра адреси;
- trireg** – модель акумулятора і програмного лічильника;
- control** – модель керуючого пристрою;
- alu** – модель АЛП.

Файл **cpu.vhd** (лістинг 8) містить структурний опис МП. Компоненти моделі МП сполучені між собою за допомогою внутрішніх сигналів і операторів **PORT MAP**.

Лістинг 8 – Файл cpu.vhd

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpulib.ALL;
ENTITY cpu IS
PORT
(
    clock      : IN  STD_LOGIC;          -- clock
    ready      : IN  STD_LOGIC;          -- ready
    reset      : IN  STD_LOGIC;          -- reset
    data :INOUT STD_LOGIC_VECTOR(15 DOWNT0 0);  -- data
    rw         : OUT STD_LOGIC;          -- rw
    vma        : OUT STD_LOGIC;          -- vma
    address    : OUT STD_LOGIC_VECTOR(15 DOWNT0 0));  -- addr
END cpu;

ARCHITECTURE cpu of cpu IS
-----
COMPONENT core
PORT
(
    reset      :IN  STD_LOGIC;
    clock      :IN  STD_LOGIC;
    command    :IN  STD_LOGIC_VECTOR(15 DOWNT0 0);
    ready      :IN  STD_LOGIC;
    aluSel     :OUT  op_type;
    regSel     :OUT  t_reg;
    PRegWr     :OUT  STD_LOGIC;
    PRegRd     :OUT  STD_LOGIC;
    addrWr     :OUT  STD_LOGIC;
    ARegRd     :OUT  STD_LOGIC;
    ARegWr     :OUT  STD_LOGIC;
    instrWr    :OUT  STD_LOGIC;
    
```

```

    regRd      :OUT STD_LOGIC;
    regWr      :OUT STD_LOGIC;
    we1_alu    :OUT STD_LOGIC;
    we2_alu    :OUT STD_LOGIC;
    rw         :OUT STD_LOGIC;
    vma        :OUT STD_LOGIC
  );
END COMPONENT;
-----
COMPONENT colcate                                  -- компонент colcate
PORT
(
  data  : IN  STD_LOGIC_VECTOR (15 DOWNT0 0);
  oper  : IN  STD_LOGIC_VECTOR (15 DOWNT0 0);
  we1   : IN  STD_LOGIC;
  we2   : IN  STD_LOGIC;
  func  : IN  op_type;
  result : OUT STD_LOGIC_VECTOR (15 DOWNT0 0)
);
END COMPONENT;
-----
COMPONENT REGARRAY                                -- компонент REGARRAY
PORT
(
  data: IN    STD_LOGIC_VECTOR (15 DOWNT0 0);
  sel : IN    t_reg;
  en  : IN    STD_LOGIC;
  clk : IN    STD_LOGIC;
  q   : OUT   STD_LOGIC_VECTOR (15 DOWNT0 0) );
END COMPONENT;
-----
COMPONENT trireg                                  -- компонент TRIREG
PORT
(
  a   : IN  STD_LOGIC_VECTOR(15 DOWNT0 0);
  clk : IN  STD_LOGIC;
  en  : IN  STD_LOGIC;
  q   : OUT STD_LOGIC_VECTOR(15 DOWNT0 0)
);
END COMPONENT;
-----
COMPONENT reg                                      -- компонент REG
PORT
(
  a   : IN  STD_LOGIC_VECTOR (15 DOWNT0 0);
  clk: IN  STD_LOGIC;
  q   : OUT STD_LOGIC_VECTOR (15 DOWNT0 0)
);
END COMPONENT;
-----
SIGNAL instrWr      : STD_LOGIC;      -- Запис в регістр команд
SIGNAL regSel       : t_reg;          -- Команда файлу операційних регістрів
SIGNAL regRd        : STD_LOGIC;      -- Читання операційного регістра

```

```

SIGNAL regWr      : STD_LOGIC;      -- Запис в операційний регістр
SIGNAL we1_alu    : STD_LOGIC;      -- Запис операнда 1 АЛП
SIGNAL we2_alu    : STD_LOGIC;      -- Запис операнда 2 АЛП
SIGNAL aluSel     : op_type;        -- Команда АЛП
SIGNAL aluout     : STD_LOGIC_VECTOR (15 DOWNTO 0); -- Результат АЛП
SIGNAL ACregRd    : STD_LOGIC;      -- Читання регістра акумулятора
SIGNAL ACregWr    : STD_LOGIC;      -- Запис в регістр акумулятора
SIGNAL PCregRd    : STD_LOGIC;      -- Читання лічильника команд
SIGNAL PCregWr    : STD_LOGIC;      -- Запис в лічильник команд
SIGNAL addrWr     : STD_LOGIC;      -- Запис в регістр адреси
SIGNAL command    : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN

```

```

-----
CONTROL : core                                -- Підключення CONTROL (CORE)
PORT MAP

```

```

(
  reset      => reset,
  clock      => clock,
  command    => command,
  ready      => ready,
  aluSel     => aluSel,
  regSel     => regSel,
  PCregWr    => PCregWr,
  PCregRd    => PCregRd,
  addrWr     => addrWr,
  ACregWr    => ACregWr,
  ACregRd    => ACregRd,
  instrWr    => instrWr,
  regRd      => regRd,
  regWr      => regWr,
  we1_alu    => we1_alu,
  we2_alu    => we2_alu,
  rw         => rw,
  vma        => vma
);

```

```

-----
ALU : calculate                                -- Підключення ALU (CALCULATE)
PORT MAP

```

```

(
  data       => data,
  oper       => data,
  we1        => we1_alu,
  we2        => we2_alu,
  func       => aluSel,
  result     => aluout
);

```

```

-----
ACC : trireg                                    -- Підключення ACC (TRIREG)
PORT MAP

```

```

(
  a          => aluout,
  clk        => ACregWr,
  en         => ACregRd,

```

```

    q          => data
  );
-----
PC : trireg                      -- Підключення PC (TRIREG)
PORT MAP
  (a          => data,
   clk        => PCregWr,
   en         => PCregRd,
   q          => data );
-----
ADDR : reg                      -- Підключення ADDR (REG)
PORT MAP
  (a          => data,
   clk        => addrWr,
   q          => address );
-----
REGS : regarray                 -- Підключення REGS (REGARRAY)
PORT MAP
  (clk        => regWr,
   data       => data,
   en         => regRd,
   q          => data,
   sel        => regSel);
-----
INSTR : reg                     -- Підключення INSTR (REG)
PORT MAP
  ( a          => data,
   clk        => instrWr,
   q          => command);
-----
END cpu;

```

На рис.2.3 наведена ілюстрація – логічна схема мікропроцесора, що створена майстром Code2Graphics CAIP Active HDL з файлу **cpu.vhd**.

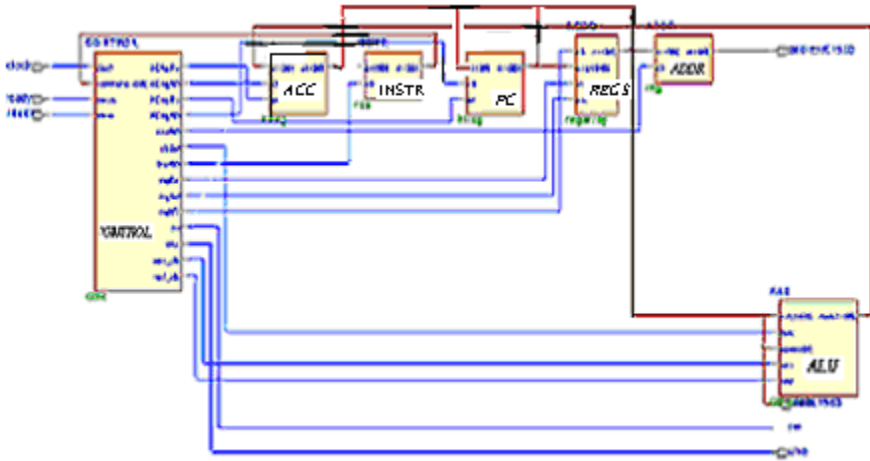


Рисунок 2.3 - Логічна схема МП

2.4.8 Файл верхнього рівня VHDL-проекту

Файл верхнього рівня VHDL-проекту (модель МП) створений в лаб. 3, але замість імітатора **icpu** треба підставити реальний **cpu** (порти в них однакові). Змінена модель МП міститься у файлі **top.vhd**.

Лістинг 8 – Файл top.vhd

 -- файл верхнього рівня проекту (система в зборі)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE work.cpu.lib.ALL;
ENTITY top IS
  PORT
  (
    reset : IN STD_LOGIC := '0'
  );
END top;
```

```
ARCHITECTURE top of top IS
  COMPONENT clk_gen
  (
    clock : OUT STD_LOGIC
  );
  -- компонент clk_gen
```

END COMPONENT;

 COMPONENT mem -- компонент MEM
 PORT

```
(
  address : IN  STD_LOGIC_VECTOR (15 DOWNTO 0);
  vma      : IN  STD_LOGIC;
  rw       : IN  STD_LOGIC;
  data     : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
  ready    : OUT STD_LOGIC
);
```

END COMPONENT;

 COMPONENT cpu -- компонент CPU
 PORT

```
(
  clock      : IN  STD_LOGIC;
  ready      : IN  STD_LOGIC;
  reset      : IN  STD_LOGIC;
  data       : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
  rw         : OUT STD_LOGIC;
  vma        : OUT STD_LOGIC;
  address    : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
);
```

END COMPONENT;

 SIGNAL clock : STD_LOGIC := '0'; -- Тактовий сигнал
 SIGNAL reset : STD_LOGIC := '0'; -- Скидання
 SIGNAL address : STD_LOGIC_VECTOR (15 DOWNTO 0); -- Шина адреси
 SIGNAL data : STD_LOGIC_VECTOR (15 DOWNTO 0); -- Шина даних
 SIGNAL vma : STD_LOGIC; -- Фіксація адреси
 SIGNAL rw : STD_LOGIC; -- Операція (0 - читання; 1 - запис)
 SIGNAL ready : STD_LOGIC; -- Підтвердження даних
 BEGIN

 g1 : clk_gen -- Підключення CLK_GEN
 PORT MAP

```
(
  clock      => clock
);
```

 m1 : mem -- Підключення MEM
 PORT MAP

```
(
  address    => address,
  vma        => vma,
  rw         => rw,
  ready      => ready,
  data       => data
);
```

 u1 : cpu -- Підключення CPU
 PORT MAP

```
(
clock      => clock,
reset      => reset,
ready      => ready,
address    => address,
rw         => rw,
vma        => vma,
data       => data
);
```

```
END top;
```

2.5 Введення моделей в систему Active-HDL

Розглянуті моделі компонентів треба ввести в систему Active-HDL. Для цього можна використати **HDL - редактор**, за допомогою якого вводиться вся модель компонента (**ENTITY** і **ARCHITECTURE**), або декларативну частину (**ENTITY**) ввести за допомогою майстра **New Source File Wizard**, як це робилось в лаб. 3.

З'єднання компонентів можна виконати за допомогою редактора блок-схем **Block Diagram Wizard**, як це робилось в лаб. 3, але значно простіше зробити це за допомогою операторів **PORT MAP**.

Модель керуючого автомата можна створити і ввести в систему Active-HDL за допомогою майстра **State Diagram** (додаток А), але простіше за допомогою редактора **HDL**.

Щоб запобігти подвійному визначенню файла верхнього рівня проекту, у вікні **Design Browser** необхідно виконати «чистку» – видалити непотрібні копії компонента **top**. Для видалення об'єкту з проекту необхідно клацнути на його піктограмі **правою** кнопкою миші і в контекстному меню вибрати команду **Delete**.

Відновіть зміст бібліотеки проекту. Спочатку, клацнувши **правою** кнопкою миші на піктограмі бібліотеки, оберіть команду **Delete simulation data** контекстного меню. Потім знову виконайте компіляцію проекту у режимі **Compile All with File Reorder**. У випадку безпомилкової трансляції бібліотека проекту буде створена знову. Вона міститиме усі необхідні данні для функціонального моделювання створеної мікропрограмної системи.

2.6 Функціональне моделювання МП

Метою функціонального моделювання є отримання часових діаграм, що описують поведінку МП під час виконання всієї програми. Функціональне моделювання МП виконується аналогічно, як у лаб.5.

На початку функціонального моделювання треба визначити *об'єкт верхнього рівня (Top-Level)* – пару описів сутності (entity) і архітектури (architecture), відносно якої це моделювання буде виконуватися. У данному випад-

ку моделюється поведінка всієї мікропрограмної системи, якій відповідає об'єкт **top (top)**. У вікні **Design Browser** розкрийте вміст компонента **top**, клацнувши символ  перед його ім'ям. Оберіть пару **top (top)** і клацніть **правою** кнопкою миші. У контекстному меню оберіть команду **Set as Top-Level** (англ. «встановити в якості файлу верхнього рівня»).

Виконайте ініціалізацію моделювання. Для цього оберіть команду **Simulation** → **Initialize Simulation** головного меню САПР. Після ініціалізації стане активною сторінка **Structure** у вікні **Design Browser**.

Відкрийте вікно редактора *часових діаграм* **Waveform Editor**. Для цього оберіть команду **File** → **New** → **Waveform**, або клацніть на кнопці  **New Waveform** панелі інструментів Active-HDL. З'явиться вікно часової діаграми **Waveform**.

Спочатку треба ввести в вікно **Waveform** сигнали вибраного компонента і задати їм відповідні стимулятори. Клацніть по імені компонента у верхній частині вікна **Design Browser** на вкладці **Structure**, внизу вікна можна побачити всі сигнали вибраного модуля. Перетягніть потрібні сигнали в вікно **Waveform**.

Клацніть **правою** кнопкою по імені сигналу у вікні **Waveform** і в контекстному меню виберіть команду **Stimulators**. З'явиться вікно **Stimulators**, де можна задавати тестові послідовності для вибраних сигналів.

Для виконання моделювання в САПР Active-HDL використовуються п'ять кнопок, розташованих на панелі інструментів. Якщо потрібно, то змініть заданий за умовчанням крок моделювання 100 ps на нове значення. Для цього клацніть на комбінованому списку і надрукуйте нове значення (рис. 2.4).

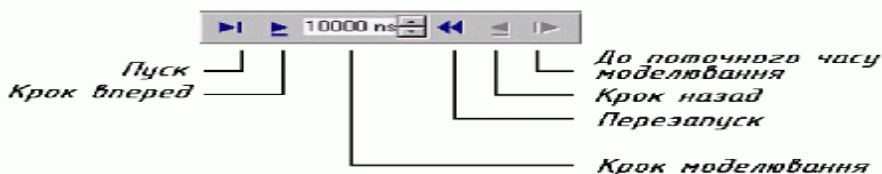


Рисунок 2.4 - Кнопки управління моделюванням

Клацнувши кнопку  **Run for**, виконайте моделювання при заданих налаштуваннях вхідних сигналів протягом першого кроку. Результати моделювання будуть відображені у вікні **Waveform Editor**. Клацаючи кнопкою **Run for**, можна продовжити моделювання в часі.

Наведіть результати моделювання послідовності станів скидання і початок виконання команди **LOADI R1**.

Наведіть результати моделювання процесу виконання команди **STORE**.

2.7 Зміст звіту

1. Мета роботи
2. VHDL – тексти всіх блоків системи з коментарями.
3. Результати моделювання окремих блоків і системи в цілому.
4. Висновки

2.8 Контрольні питання

1. Поясніть принцип роботи мікропроцесора, що проектується.
2. Що таке інформаційний та керуючий тракти?
3. З яких файлів складається VHDL- проект ЦС? Для чого потрібно створювати робоче середовище (workspace)? Яку функцію у VHDL- проекті виконує файл верхнього рівня ієрархії? Для чого в описі ЦС використовують модуль entity? Модуль architecture?
4. Побудуйте граф-схему алгоритму програми.
5. Поясніть послідовність виконання машинних команд.
6. Що таке алфавіт станів керуючого автомата? Назвіть їх.
7. Що міститься у бібліотеці STD_LOGIC_unsigned? Яким чином підключаються бібліотеки до програми мовою VHDL?
8. Які засоби редагування має редактор HDL? Поясніть техніку їх використання.
9. Які функції виконує майстер Language Assistant? Поясніть техніку його використання.
10. Які засоби редагування має редактор BDE? Поясніть техніку їх використання.
11. Яка різниця між висхідною та низхідною стратегіями проектування? Поясніть техніку реалізації обох стратегій в редакторі BDE.
12. Які сигнали називаються глобальними? Як задається глобальний сигнал в редакторі BDE?
13. Які засоби редагування має редактор FSM? Поясніть техніку їх використання. Чи можна в САПР Active-HDL розробляти граф переходів кінцевого автомата, що не має тактового входу?
14. Яким чином задаються параметри кінцевого автомата? Яка різниця між кінцевими автоматами Мура і Мілі?

ЛІТЕРАТУРА

1. Перельройзен Е.З. Проектируем на VHDL – М.: СОЛОН-Пресс, 2004.- 448с.
2. Сергиенко А.М. VHDL для проектирования вычислительных устройств.- К.: ЧП "Корнейчук", ООО "ТИЛ ДС", 2003. – 208с.
3. Проектирование цифровых систем с использованием языка VHDL: Уч. пособие/ В.В. Семенец, И.В. Хаханова, В.И. Хаханов.– Харьков: ХНУРЭ, 2003.– 492 с.
4. Методичні вказівки до виконання курсового проекту з курсу "Теорія і проектування комп'ютерних систем та мереж" для підготовки спеціалістів та магістрів на базі напрямку "Комп'ютерна інженерія" / Укл. Дунець Б.Р., Мельник А.О., Троценко В.В. – Львів: Видавництво ДУ "Львівська політехніка", 1999. – 25 с.
5. Методичні вказівки до лабораторних робіт з дисципліни “Архітектура ЕОМ” для студентів спеціальностей 8.091 501 ”Комп'ютерні системи та мережі” та 7.091 503 ”Спеціалізовані комп'ютерні системи” всіх форм навчання. Частина 2. /Укл. А.К. Тімовський, О.О.Голдобін. - Запоріжжя: ЗНТУ, 2004. – 50 с.
6. Методичні вказівки до лабораторних робіт з дисципліни “Основи автоматизованого проектування засобів обчислювальної техніки” для студентів спеціальностей 8.091 501 ”Комп'ютерні системи та мережі” та 7.091 503 ”Спеціалізовані комп'ютерні системи” всіх форм навчання. Частина 2. /Укл. А.К.Тімовський, О.О.Голдобін. - Запоріжжя: ЗНТУ, 2004. – 58 с.
7. Методичні вказівки до лабораторних та самостійних робіт з дисципліни “Спеціалізовані архітектури ЕОМ” для студентів спеціальності 7.091 503 ”Спеціалізовані комп'ютерні системи” всіх форм навчання /Укл. С.С. Грушко, О.О.Голдобін. - Запоріжжя: ЗНТУ, 2004. – 50 с.
8. А.К. Тімовський, , О.О.Голдобін Методичні вказівки до лабораторних робіт з дисципліни “Автоматизація проектування комп'ютерних систем”.– ЗДТУ, 2009

Додаток А

Створення моделі керуючого автомата за допомогою редактора State Diagram

1. Додамо до створеного проекту новий файл керуючого автомата, для чого клацніть двічі мишкою по команді **Add New File** у вікні **Design Browser**.

У вікні **Add New File**, що з'явилося, виберіть **Empty Files**, а в ньому об'єкт – **State Diagram** і введіть ім'я файлу опису графа переходів кінцевого автомата **control** в полі **Name**. Клацніть **OK**.

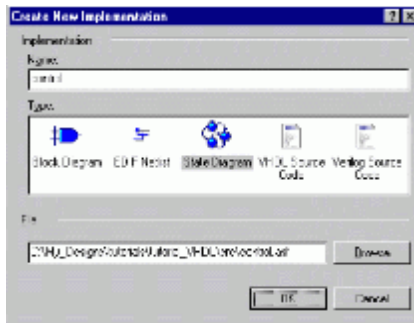


Рисунок А.1- Вікно **Add New File – Empty Files**

2. В проєкті повинен з'явитися новий файл шаблона опису графа переходів автомата **control.asf**. Це можна побачити у вікні **Design Browser**.

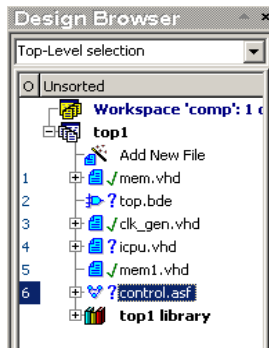


Рисунок А.2– Додавання файлу control.asf в проєкт

3. **Перейдемо** до введення графу переходів кінцевого автомата. Введення графа і визначення його основних **налаштувань** виконується у вікні редактора FSM. На початку роботи створимо порти – сигнали, які грають роль *логічних умов* (ЛУ) переходів. Для введення портів використовується інструмент  **Input Port (FSM → Port → )** для вхідних портів і інструмент  **Output Port** для вихідних. Розмістіть 4 вхідних порти у верхній декларативній частині робочого простору документа, відкритого у вікні редактора FSM, зліва ,а справа - 13 вихідних портів.

4. Змініть стандартні імена портів. Для цього спочатку перемкніться в режим роботи **Select**, а потім, послідовно перейменуйте порти. Для того, щоб розпочати редагування імені порту, необхідно двічі клацнути на його імені. Вхідні порти повинні називатися: **reset**, **clock**, **command**, **ready**, а вихідні порти **aluSel**, **regSel**, **PCregWr**, **addrWr**, **ACregWr**, **ACregRd**, **instrWr**, **regRd**, **regWr**, **we1_alu**, **we2_alu**, **rw**, **vma**.

Збережіть параметри налаштувань, використовуючи комбінацію "гарячих" клавіш <CTRL+S>.

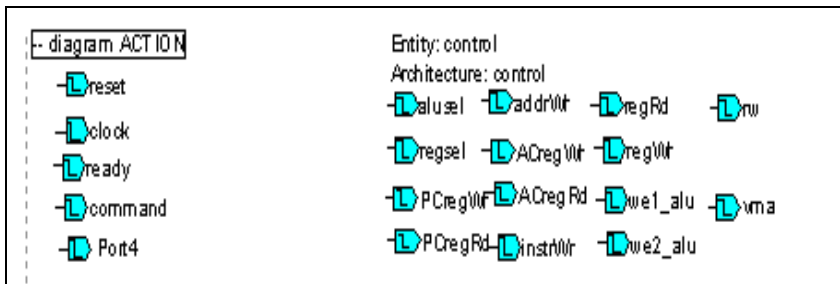


Рисунок А.3 – Оголошення вхідних та вихідних портів керуючого автомата

5. Вхід **clock** слугує сигналом синхронізації. Встановимо режим роботи цього входу. Для цього клацніть **правою** кнопкою миші по його зображенню і виберіть команду **Properties** контекстного меню.

1. У вікні **Port Properties** відзначте налаштування **Clock** і підтвердить це, клацнувши ОК. Сигнал **Clock** тепер помічений відповідною піктограмою.

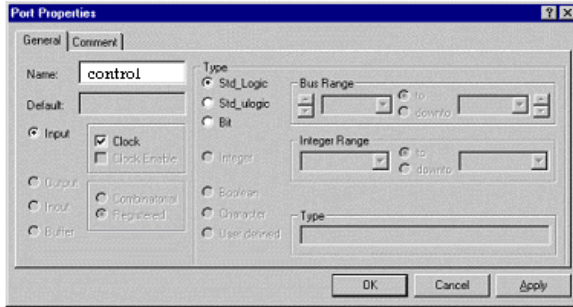


Рисунок А.4 - Зміна налаштувань порту

7. Таблиця переходів приведена в таблиці 1.14. Це автомат Мура, що має багато станів. За допомогою інструменту  **State** розмістіть вершини-стани в нижній частині вікна редактора FSM (рис. А.5). Для зміни положення вершини треба навести курсор на вершину і, утримуючи ліву кнопку мишки, перетягнути вершину на нове місце.

8. Для вказівки переходів в автоматі використовують інструмент  **Transition**. Встановивши курсор миші на стані передавача, натискайте ліву кнопку і перетягніть лінію-перехід в стан приймача. Утворюється **напряме-на** гілка графа переходів. В результаті повинен вийти граф автомата, зображений на рис. А.5 (тільки для однієї команди ADD).

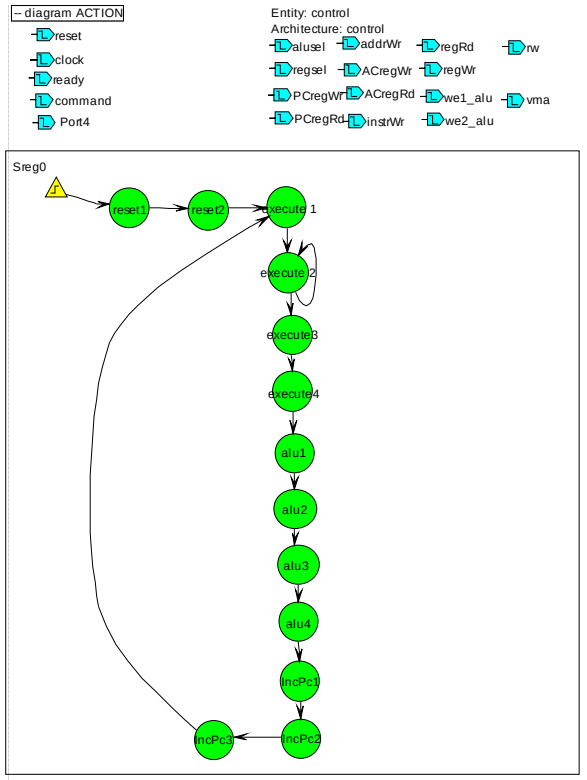


Рисунок А.5 – Розміщення станів та переходів керуючого автомата

9. Для запуску/скидання керуючого автомата передбачається використовувати сигнал асинхронного скидання **reset**. Для його визначення використовується спеціальний інструмент –  **Reset**. Розмістіть піктограму скидання недалеко від початкового стану автомата і вкажіть відповідний перехід (рис. А.5).

10. Дайте імена станам автомата відповідно до табл. 2.6.

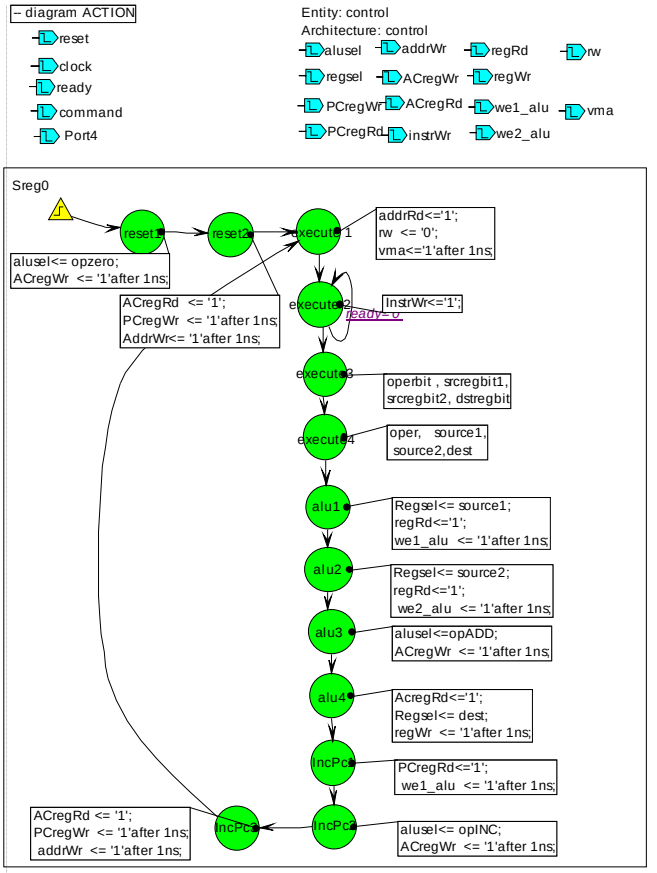


Рисунок А.6 - Граф переходів керуючого автомата для однієї команди ADD

11. Визначимо умови переходів автомата. Щоб це зробити, треба вибрати інструмент  **Condition**, вказати потрібну гілку графа і в полі, що з'явиться, надрукувати формулу переходу. Задайте умови переходів, як показано на рис. А.1.

12. Для кожного стану треба вказати значення вихідних сигналів. Для того, щоб виконати цю операцію, виберіть інструмент  **State Action**, вкажіть стан і в полі, що з'явилося, надрукуйте відповідну формулу (див. табл.1.14). Результат роботи показаний на рис. А.1.

13. Аналогічно добудовуємо граfi інших команд. Результат роботи показаний на рисунку А.7.

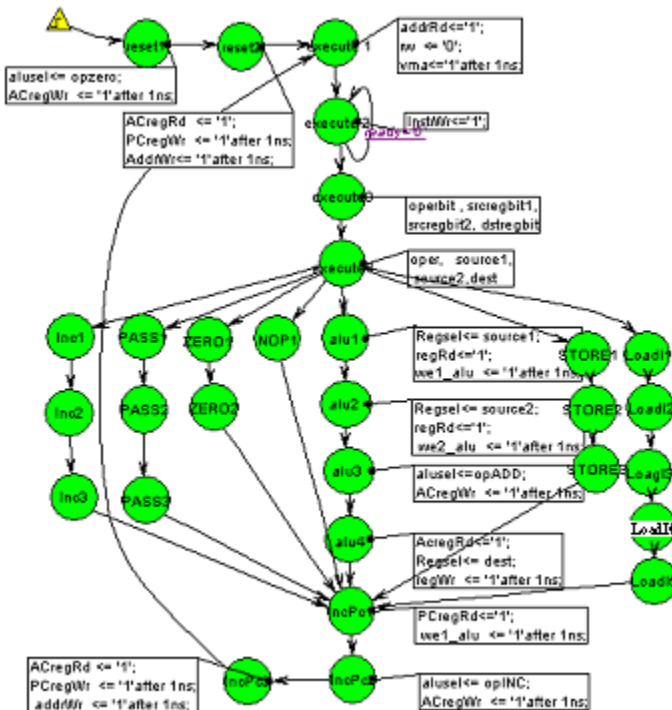


Рисунок А.7 - Граф переходів керуючого автомата

14. Щоб вилучити помилковий надпис або лінію, треба його виділити за допомогою **Select** і натиснути кнопку **Delete**. Введення графа автомата закінчене. Залишилося задати тип сигналу запуску/скидання автомата. Клацніть **правою** кнопкою миші у вікні редактора і в контекстному меню виберіть команду **Properties**.

15. У вікні **Machine Properties** знайдіть вкладку **RESET** і змініть тип сигналу запуску/скидання автомата з **Synchronous** на **Asynchronous**. Підтвердіть настройки автомата, клацнувши **OK**. Збережіть файл **control.asf**, використовуючи комбінацію клавіш <CTRL+S>.

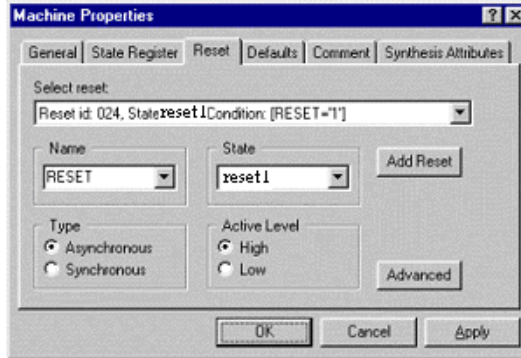


Рисунок А.8 - Налаштування керуючого автомата

11. Клацнувши кнопку  **Compile**, виконайте трансляцію опису графа переходів автомата в текст мовою VHDL. Про те, що трансляція пройшла успішно, з'явиться відповідне повідомлення у вікні **Console**, і у вікні **Design Browser** навпроти файлу **control.asf** буде поміщена зелена піктограма.